# MICRO

## THE 6502 JOURNAL

SAMPLE MACHINE LANGUAGE PROGRAM
AS INPUTTED FROM THE KEYBOARD

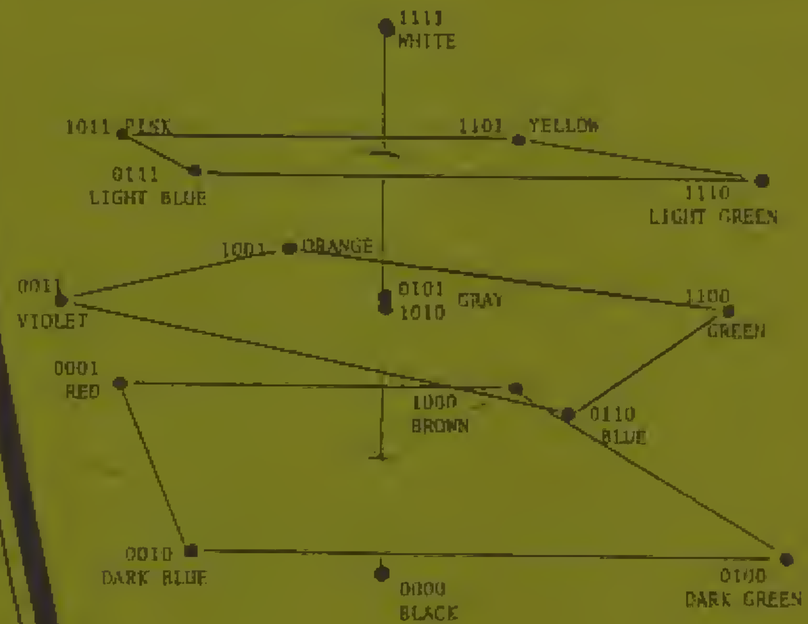
```
? ORG 826
? LDAIM 102
? LDXIM 0
? STAX 32768
? INX
? BEQ 3
? JMP 830
? NOP
? NOP
? STAX 33024
? INX
? BEQ 3
? JMP 841
? BRK
? END
```

A Simple 6502
Assembler for the PET
by Michael J. McCann

Complete Listings

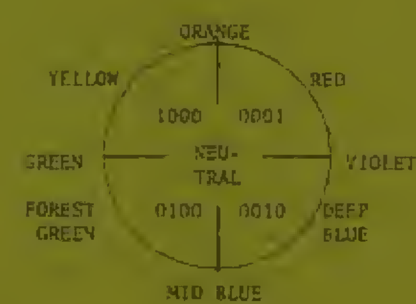
SAMPLE MACHINE LANGUAGE PROGRAM LISTING

```
826 033A  A9 66     LDAIM   102
828 033C  A2 00     LDXIM   0
830 033E  9D 00 80  STAX    32768
833 0341  E8        INX
834 0342  F0 03     BEQ     3
836 0344  4C 3E 03  JMP     830
839 0347  EA        NOP
840 0348  EA        NOP
841 0349  9D 00 81  STAX    33024
844 034C  E8        INX
845 034D  F0 03     BEQ     3
847 034F  4C 49 03  JMP     841
850 0352  00        BRK
```




Brown and White and Colored All Over
by Richard F. Suitor

Understanding your Apple's Color




NO 6 Aug - Sept 1978 $1.50

MICRO

AUGUST/SEPTEMBER 1978

ISSUE NUMBER SIX



Advertisers Index



MICRO is published bi-monthly by The COMPUTERIST, Inc., 56 Central Square, Chelmsford, MA 01824. Robert M. Tripp, Editor/Publisher. Controlled Circulation postage paid at Chelmsford, Massachusetts.

Single Copy: $1.50 Annual Subscription: $6.00 (6 issues) in USA

## IN THIS ISSUE ...

There were so many good articles submitted for this issue of MICRO that we have had to modify the format slightly to make more room. Most of the MICRO material has been reduced to approximately two-thirds its old size, providing about 50% more space per page. While this does make type smaller, it is still very readable. Some material, in particular program listings, were left full size. This new format will permit us to print a lot more material without increasing the cost of printing.

How do you get hardcopy from a PET? You could wait until Commodore comes out with a printer. Or you could buy one of the PET/RS232 adapters. Or you can use the techniques and software that are presented in "Design of a PET/TTY Interface" to quickly and cheaply use a standard TTY as a PET printer. The article by Charles R. Husbands provides both the hardware and the software required.

If you have wondered about how the characters formed on your Apple II, read "Shaping Up Your Apple" by Michael Faraday. In addition to explaining how the mechanism works, a couple of tables make it easy to make your own adaptations.

Now that STARWARS is back at your local drive-in, it seemed appropriate to print a short program by Andrew H. Eliason which presents the "Apple II Starwars Theme" - sounds of the main battle scene played on your Apple. While this program may give you some insight into the operation of your Apple, it is really included just for fun.

On a more serious vein, in spite of its humorous title, "Apple Pi" shows how to use BASIC to calculate mathematical functions. Robert J. Bishop presents the history of calculating Pi, and then provides a program which, given forty hours, can calculate the value of Pi to 1000 decimal places. In case you do not want to run the program yourself, the results of his run are printed. It might be a challenge to someone to write the equivalent code in assembly language and see how long it takes to run.

One of the most constant complaints of PET owners is the lack of support for assembly level programming on the PET, in spite of promises by Commodore for a ROM or tape of a machine code monitor. This will be partially alleviated by "A Simple 6502 Assembler for the PET" by Michael J. McCann, complete in this issue. The package presented here consists of the assembler, a save on tape routine, a load from tape routine, and a disassembler to produce listings. Two errors in the listing were discovered after that portion of MICRO was printed, so please make the following changes in the listings:

```
190  IF VAL(A$)<1 OR VAL(A$)>6 GOTO 180

15020  IF LEN(A$)=3 THEN MN$=A$:OP=0:RETURN
```

Since the "BASIC 6502 Disassembler" written by Michael for the last issue of MICRO was, with very minor modification, capable of running on an Apple as well as a PET, the assembler portion of this program is probably also modifyable for the Apple. The exercise is left for the reader, as the math books are fond of saying.

Part III of the MICRO Software Catalog has eight entries covering a wide variety of software and systems. These range from a program to punch readable leader of a paper tape to FOCAL - a DEC high-level language similar to BASIC.

There is a "Call for Information" in regards to a MICRO Hardware Catalog which we hope to start carrying in the next issue. If you have hardware of interest to the 6502 community, then follow the instructions and submit your stuff.

A rather neat program which serves as "A Debugging Aid for the KIM-1", written by Albert Gaspar, provides some good support for the KIM-1 and resides totally in the "extra memory" from 1780 to 17E6. Four basic operations are given:

Insert BREAK points, MOVE blocks of data in memory, calculate BRANCH offsets, and CONTINUE execution of the program.

The program is very tightly coded and shows some ways to really pack your code.

The series on "6502 Interfacing for Beginners" continues with "Address Decoding II". This series, which began last issue and is written by Marvin L. De Jong, shows the novice how the microcomputer works via simple hardware and software projects.

One of the most obvious features of the Apple II is its color capabilities. The article "Brown and White and Colored All Over" by Richard F. Suitor explains in some detail the theory behind the color of the Apple. He also provides a few simple BASIC programs to allow the user to do some experimenting with color.

Part V of the "6502 Bibliography" by William Dial covers entries 335 through 360. Due to the "explosion" of material being written about the 6502, some changes have had to be made in the organization and content of the bibliography. Straight advertisements will no longer be referenced or will material contained in flyers. Minor articles in relatively obscure magazines may be omitted. And, where a single issue of a magazine has a lot of articles of interest, the individual references will be combined under one general magazine reference.

"Programming a Micro-Computer: 6502"a book by Caxton C. Foster, is reviewed by James R. Witt, Jr.

Cal E. Merritt discusses the "PET Composite Video Output", showing how it works and how to connect up to it. Karl E. Quosig whows how to get "Power from the PET", a method of getting +5V from your PET.

A "Classified Index: MICRO 1-6" lists all of the major articles and advertisements from the first volume/year of MICRO. Material is classified as General, KIM-1, Apple, PET, or Ads.

A very useful utility package is presented by Richard F. Suitor in "Apple Integer BASIC Subroutine Pack and Load". The assembly level program, which is presented in its entirety, permits the user to simply Pack and save his machine code on tape and the Load and unpack it.

"A Partial List of PET Scratch Pad Memory" is printed on the back cover as a reference guide for PET owners. This material was prepared by Gary A. Creighton, and should make using and understanding your PET much easier.

## Apple Peelings

[Excerpts from a letter by Donald C. Scouten to the Editor, EDN, regarding the Apple/PIA stuff.]

"The difficulty in using PIA's and VIA's on the Apple II arises because of the way the Apple decodes the I/O select (pin 1) and device select (pin 41). These are activated only during phase 2 of a cycle that addresses the particular connector under consideration. Thus, if these selects are used ... to activate the CS (or not CS) on a PIA, the enable pin (pin 25) and the CS go active almost simultaneously. However the data sheets clearly require a 180 nsec setup time for the CS before the enable becomes active. This setup time is normally available on 6502 bus since the addresses are guaranteed to be valid 300 nsec into phase 1 (and thus your circuit worked on a KIM)... It is, however, clearly impossible to use the internal Apple decoding and satisfy the PIA ... requirement of 180 nsec setup time.

The above problem should not be interpreted as a defect in the Apple II since it is a self consistant system and I/O ports can easily be added if desired.

My solution was to build a simple address decoder on my I/O board that uses the address lines instead of the selects. Thus the CS of the VIA is activated with sufficient setup time and the VIA works properly."

A note from Paul Farmer of Microproducts, 1024 17th St., Hermosa Beach, CA 90254, suggests using three buffers in series on a CMOS 4050 IC chip. Either phase 0 or phase 2 can be used as the input with enough delay for the setup of a PIA or VIA.

## PET Droppings

A new idea in magazines: CURSOR (tm) MAGAZINE is a monthly cassette of programs for the PET. You get five programs per month on cassette via 1st class mail. At $24.00 per year (12 issues),the cost per program is $.40 cents each. Of course, the actual value of the programs depends on their value to you. Write CURSOR, P.O. Box 550, Goleta, CA, 93017 for info or call 805/967-0905.

Mark Zimmerman, 619 Woodland Drive, Sierra Madre CA 91024 write about the LIFE game edges:

"If one copies the top and bottom edges of the screen (& left & right edges) to opposite sides, then simply applying the LIFE algorithm to the central (omitting extreme edges) arena gives correct wrap-around (toroidal) edge structure. Example:

```
                      L I J K L I
  A B C D             D A B C D A
  E F G H             H E F G H E
  I J K L             L I J K L I
                      D A B C D A
```

## Kim Klippings

The San Fernando Valley KIM-1 Users Club is off and running, according to a report from Jim Zuber. Meetings will be held the second Wed. of each month at 7:30 pm. Until another place can be found, meetings will be held in Jim's apartment: 20224 Cohassett #16, Canoga Park, CA 91306. Phone for inof: 213/341-1610.

Michael Chibnik of 10445 Canoga Ave. Chatsworth CA 91311, had a few comments about Microsoft BASIC for the KIM: "I didn't get enough information on the peripherals that were used. A note about Microsoft BASIC is that most of the people who had bought it (in the above club) did not like the fact that the code for the interpreter is self modifying in many places and that it is not PROMable." [Editor: Someone reported that they had asked Johnson Computer about the PROM-ability of the Microsoft BASIC and was told that it is PROMable. Does anyone have any hard info on this subject?]

Robert Ford Denison, RD 5 Teeter Road, Ithaca, NY 14850 has developed a resident symbolic 6502 assembler which runs in 3K (4K recommended) and uses a "Qwerty" keyboard for input and the KIM display for output. To test it he is "offering a free 'sneak preview' of the assembler to a small group of 6502 users ... (since he) would appreciate comments on any parts of the documentation that are not perfectly clear. Write him for further information.

## General Garbage

You might want to write to Robert Elliott Purser at P.O. Box 466, El Dorado, CA 95623 and request a copy of his "World's Second Most Incomplete Software List for PET, Radio Shack, Apple & Sol"

## MICROBES

Applayer Music Interpreter, Suitor, 5:29:

```
5:30  0A20-  82 20 0B
5:31  0A00:  83 90 0F 83 90 0F FF
      0F18:  1C 1A 18 1A 91 1C 38 18
      0F50:  81 55 55 55 FF
      0F58:  81 05 05 05 FF
      0F90:  83 58 0F D4 B0 83 50 0F 83
      0810:  48 02 28 02 08 02 E8 01
```

These problems are in the music and tone table, and were caused by the 8's on his TTY looking very much like 0's. Make the changes and the music will probably sound better.

A BASIC 6502 Disassembler for Apple and PET, McCann, 5:25:

```
5:26  3020:  DC=IB:GOSUB 1000
5:27  6000:  ASL should be ASLZ
      6100:  CLC should be CLI
      6120:  JMI should be JMPI
      6250:  CPX should be CPXZ
```

D/A and A/D Conversion Using the KIM-1, De Jong, 2:11: IC should be labeled "1408" and pin 14 should have 1.5K resistor to +5, while pin 13 goes directly to +5V (check spec sheets on 1408 to be absolutely sure of connections).

0308 4C 0403 should be 4C 05 03

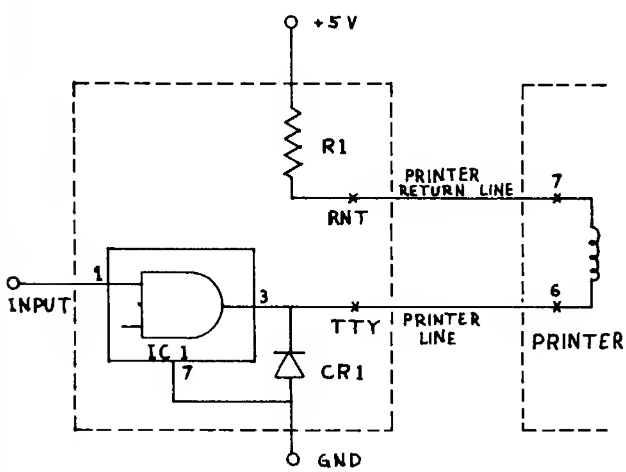
# DESIGN OF A PET/TTY INTERFACE

Charles R. Husbands
24 Blackhorse Drive
Acton, MA 01720

With the recent acquisition of a PET Computer one of the facilities that was immediately needed was a method of obtaining hard copy listings of programs under development. In addition to the PET I had an ASR 33 Teletype Unit available which had been interfaced to my KIM-1. This article describes the hardware interface and associated software necessary to use the ASR 33 TTY as a printing facility for the PET. An important design goal for the interface was to develop the software to remain resident in the computer in such a manner that the program under development could be loaded, run and listed without disturbing the listing program.

## The Interface Circuit

Figure 1 shows the 20 ma current loop circuit required to interface the ASR 33 to the PET. The circuit consists of an open collector NAND gate to provide the proper buffering, a diode and a pull up resistor. The completed circuit was built on a small perforated board. The PET supplies power and ground to the interface board from the second Cassette Interface. The input signal is delivered from PA0 on the PET parallel user port. The interface board is connected to the teletype by means of the PRINTER and PRINTER RETURN lines. These lines attach to terminals 6 and 7 respectively on the ASR 33.




Parts List

| | | |
|---|---|---|
| IC1 | 7438 | Quad 2 Input NAND Open Collector |
| CR1 | 1N4001 | 1A 50V Diode |
| R1 | 150 ohm | 1/2 Watt Resistor |

Figure 1.

A fairly simple circuit for buffering the control signal from the PET Computer and converting that signal to a current level capable of driving the printer mechanism on an ASR 33 TTY Unit.

## Program Design

In order to allow the listing program to remain resident in the machine to list other programs under development, the program was written in machine language to be stored in Tape Buffer 2. Figure 2 shows a simple memory map of the PET random access memory allocations. Without a second tape cassette unit, a memory buffer of 198 bytes is available. When another program is loaded from tape or the NEW instruction is executed the operating system zeros out memory locations 1024 and above. However, it leaves the memory locations below 1024 undisturbed. To execute a machine language program the USR instruction must be called. The USR command uses a pair of memory location pointers stored in memory locations 1 and 2 to extablish the first location in machine language code to be processed. Locations 1 and 2 are not modified by the loading of a program from tape or the execution of the NEW instruction.

8192 ................................ $1200

Program Storage

1024 ................................ $0500

Tape Buffer 2

826 ................................ $033A

Tape Buffer 1

634 ................................ $027A

BASIC and Operating System Working Space

2 ................................ $0002

USR Control Pointers

0 ................................ $0000

Figure 2.

A Map of the PET Random Access Memory Space. The Listing Program resides in machine language in Tape Buffer 2.

A flow diagram of the Listing Algorithm is shown in Figure 3. The program after proper initiation examines the first character of the third line in the display for a value corresponding to the letter "R". It is the letter R appearing in the first display column which is used by the Listing Program to exit the listing algorithm and return control of the program to the calling routine. The R in the first column would normally correspond to the READY displayed by the computer at the end of a requested listing block or at the completion of an executed RUN. If the character in the first column is anything but an R the program executes a carriage return and then a line feed. The program examines the next displayed character and translates it from display format to ASCII format. The subroutine PRINT is then called.

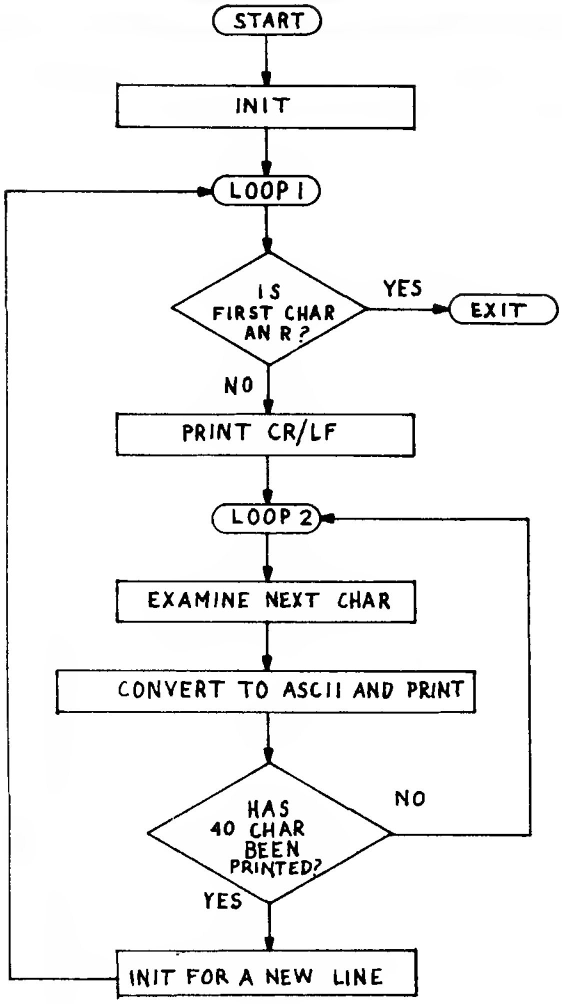



Figure 3.

A general listing algorithm for use with the TTY Listing Program. The software control of the output port is done in the PRINT subroutine.

The subroutine PRINT* is a machine language program which times out the proper serial bit pattern to the TTY to execute the printing of the designated letter. After each character is printed a counter is incremented and tested to determine if the 40 character line has been completed. If 40 characters have not been printed the next display character is examined. At the end of each line the first character of the next line is examined for an R before a carriage return and line feed is executed.

A listing of the program in BASIC format is shown in Listing 1. The program was originally hand assembled in 6502 machine language. The machine language program was then converted from hexadecimal to decimal and formatted as a series of POKE instructions. The machine language memory address pointers were also POKED into locations 1 and 2 by the BASIC program. The printout appearing in Listing 1 was produced on the authors TTY using the Listing Program.

* The PRINT subroutine is a modified version of the "PRINT 1 CHAR" program developed by MOS Technology for the KIM-1.

## Using the Listing Program

The program as shown in Listing 1 is loaded into the machine in the normal manner. A RUN command is then executed and the program will be POKED in machine format into Tape Buffer 2. The BASIC program to be listed is then loaded into the machine. The LIST-N instruction is then executed to allow the operator to preview the initial lines of code. When the operator is satisfied with the 15 to 18 lines of code to be printed, as displayed on the screen, the command X=USR(R) is entered and the RETURN key is depressed. The USR instruction transfers control to the machine language code located at the address specified by memory locations 1 and 2.

The teletype printer will then print the display on the PET CRT from the beginning of display line 3 to the word READY. The operator then uses the LIST M-X command to preview the next series of lines to be printed. It should be noted that the PET listing format leaves a blank line between the last line number selected and the READY response if the last line requested is not the last line in the program. The preview function allows the operator to block out the lines to be printed regardless of the line numbering technique employed when the program was composed. If the program being listed has an R in column 1 due to a line length in excess of 40 characters, the operator must take some action to remove this condition before executing the listing of that portion of the program.

## Conclusions and Recommendations

The hardware and software illustrated in this article can be used to permit the listing of programs and recording the results of program runs on a conventional TTY unit. In using the program to print the results of computer runs it should be noted that the results should be formatted to begin on the third line of the display. An improved version of this program could be designed to look ahead when an R was discovered to extablish if an RE or REA string was present. As only 3 bytes were not used in Tape Buffer 2 in writing this program, that feature could not be included. Additional space could be freed if the program was redesigned to use the parallel to serial conversion facility available with the 6522 VIA output port. Using this facility the 90 bytes required to do the conversion from parallel to serial and timing out this information could be greatly reduced.

Listing 1.

A listing of the PET Listing Program as printed on the author's TTY unit. The program was hand assembled in 6502 language then converted to decimal format and entered as a series of BASIC "POKE" instructions. When executed the program will reside in Tape Buffer 2 in machine code format.

```
1 REM***TELETYPE LISTING ROUTINE******
2 REM    CHARLES R. HUSBANDS
3 REM
4 REM THIS PROGRAM LISTS THE DATA
5 REM APPEARING ON THE SCREEN IN
6 REM SERIAL TELETYPE FORMAT. THE
7 REM PROGRAM IS STORED IN MACHINE
8 REM CODE IN TAPE BUFFER #2. THE
9 REM PROGRAM IS EXECUTED USING "USR".
10 POKE(01),58
20 POKE(02),03
29 REM..INIT...INITALIZE VARIABLES
30 POKE(826),169
40 POKE(827),00
50 POKE(828),141
60 POKE(829),251
70 POKE(830),03
80 POKE(831),170

88 REM..LOOP1..TEST FIRST CHAR ON EACH
89 REM  LINE FOR AN "R".
90 POKE(832),189
100 POKE(833),80
110 POKE(834),128
150 POKE(835),201
160 POKE(836),18
170 POKE(637),240
180 POKE(838),83
189 REM..LOOP3..PRINT CR/LF
190 POKE(839),169
200 POKE(840),13
210 POKE(841),141
220 POKE(842),255
230 POKE(843),03
240 POKE(844),32
250 POKE(845),166
260 POKE(846),03

270 POKE(847),169
280 POKE(848),10
290 POKE(849),141
300 POKE(850),255
310 POKE(851),03
320 POKE(852),32
330 POKE(853),166
340 POKE(854),03
348 REM..LOOP2..EXAMINE AND PRINT THE
349 REM  OTHER CHARACTERS ON THE LINE.
350 POKE(855),189
360 POKE(856),80
370 POKE(857),128
380 POKE(858),141
390 POKE(859),252
400 POKE(860),03
410 POKE(861),56
420 POKE(862),233
430 POKE(863),32
440 POKE(864),48
450 POKE(865),12
460 POKE(866),173
470 POKE(867),252
480 POKE(868),03
490 POKE(869),141
500 POKE(870),255
510 POKE(871),03
520 POKE(872),32
530 POKE(873),166
540 POKE(874),03
550 POKE(875),76
560 POKE(876),122
570 POKE(877),03
579 REM..ALPHA..PRINT ALPHABETIC CHAR
580 POKE(878),173
580 POKE(878),173
590 POKE(879),252
600 POKE(880),03
610 POKE(881),24
620 POKE(882),105
630 POKE(883),64
640 POKE(884),141
650 POKE(885),255
660 POKE(886),03
670 POKE(887),32
680 POKE(888),166

690 POKE(889),03
698 REM..CLNUP..COUNT CHARACTERS AND
699 REM  TEST FOR END OF LINE.
700 POKE(890),238
710 POKE(891),251
720 POKE(892),03
730 POKE(893),173
740 POKE(894),251
750 POKE(895),03
760 POKE(896),201
770 POKE(897),40
780 POKE(898),240
790 POKE(899),13
800 POKE(900),232
810 POKE(901),138
820 POKE(902),208
830 POKE(903),06
840 POKE(904),238
850 POKE(905),89
860 POKE(906),03
861 POKE(907),238
862 POKE(908),66
863 POKE(909),03
870 POKE(910),76
880 POKE(911),87
890 POKE(912),03
899 REM..NEWL..INITALIZES NEW LINE.
900 POKE(913),169
910 POKE(914),00
911 POKE(915),141
912 POKE(916),251
913 POKE(917),03
914 POKE(918),232

917 POKE(919),76
918 POKE(920),64
919 POKE(921),03
920 REM..FINDR..PROGRAM COMES HERE IF
921 REM  AN "R" IS FOUND IN 1ST COLM.
921 POKE(922),169
922 POKE(922),169
923 POKE(923),128
924 POKE(924),141
925 POKE(925),66
926 POKE(926),03
927 POKE(927),141
928 POKE(928),89
929 POKE(929),03
930 POKE(930),96
```

```
949 REM..PRINT..THIS SUBROUTINE PRINTS
950 REM  THE CHARACTER IN TTY FORMAT.
960 POKE(934),169
961 POKE(935),255
962 POKE(936),141
963 POKE(937),67
964 POKE(938),232
965 POKE(939),173
966 POKE(940),255
970 POKE(941),03
980 POKE(942),141
990 POKE(943),252
1000 POKE(944),03
1010 POKE(945),142
1020 POKE(946),253
1030 POKE(947),03
1040 POKE(948),32
1050 POKE(949),230
1060 POKE(950),03
1070 POKE(951),169
1080 POKE(952),79
1090 POKE(953),232
1100 POKE(954),41
1110 POKE(955),254
1120 POKE(956),141
1130 POKE(957),79
1140 POKE(958),232
1150 POKE(959),32
1160 POKE(960),230
1170 POKE(961),03
1180 POKE(962),162
1190 POKE(963),08
1199 REM..OUT1
1200 POKE(964),173

1210 POKE(965),79
1220 POKE(966),232
1230 POKE(967),41
1240 POKE(968),254
1250 POKE(969),78
1260 POKE(970),252
1270 POKE(971),03
1280 POKE(972),105
1290 POKE(973),00
1300 POKE(974),141
1310 POKE(975),79
1320 POKE(976),232
1330 POKE(977),32
1340 POKE(978),230
1350 POKE(979),03

1360 POKE(980),202
1370 POKE(981),208
1380 POKE(982),237
1390 POKE(983),173
1400 POKE(984),79
1410 POKE(985),232
1420 POKE(986),09
1430 POKE(987),01
1440 POKE(988),141
1450 POKE(989),79
1460 POKE(990),232
1470 POKE(991),32
1480 POKE(992),230
1490 POKE(993),03
1500 POKE(994),174
1510 POKE(995),253
1520 POKE(996),03
1530 POKE(997),96
1539 REM..DELAY
1540 POKE(998),169
1550 POKE(999),02
1560 POKE(1000),141
1570 POKE(1001),254
1580 POKE(1002),03
1590 POKE(1003),169
1600 POKE(1004),82
1609 REM..DE2
1610 POKE(1005),56
1619 REM..DE4
1620 POKE(1006),233

1630 POKE(1007),01
1640 POKE(1008),176
1650 POKE(1009),03
1660 POKE(1010),206
1670 POKE(1011),254
1680 POKE(1012),03
1689 REM..DE3
1690 POKE(1013),172
1700 POKE(1014),254
1710 POKE(1015),03
1720 POKE(1016),16
1730 POKE(1017),243
1740 POKE(1018),96
1750 REM..COUNT(1019)
1760 REM..CHAR (1020)
1770 REM..TMPX (1021)
1780 REM..TIMH (1022)
1790 REM..PCHAR(1023)
1800 END
```

| LABEL | OP | FIELD | LOC | OP | F1 | F2 |
|---|---|---|---|---|---|---|
| INIT | LDA | #0 | 826 | 169 | 00 | |
| | STA | COUNT | 828 | 141 | 251 | 03 |
| | TAX | | 831 | 170 | | |
| LOOP1 | LDA | 32848,X | 832 | 189 | 80 | 128 |
| | CMP | #18 | 835 | 201 | 18 | |
| | BEQ | FINDR | 837 | 240 | 83 | |
| LOOP3 | LDA | #0D | 839 | 169 | 13 | |
| | STA | PCHAR | 841 | 141 | 255 | 03 |
| | JSR | PRINT | 844 | 32 | 166 | 03 |
| | LDA | #0A | 847 | 169 | 10 | |
| | STA | PCHAR | 849 | 141 | 255 | 03 |
| | JSR | PRINT | 852 | 32 | 166 | 03 |
| LOOP2 | LDA | 32848,X | 855 | 189 | 80 | 128 |
| | STA | CHAR | 858 | 141 | 252 | 03 |
| | SEC | | 861 | 56 | | |
| | SBC | #20 | 862 | 233 | 32 | |
| | BMI | ALPHA | 864 | 48 | 12 | |
| | LDA | CHAR | 866 | 173 | 252 | 03 |
| | STA | PCHAR | 869 | 141 | 255 | 03 |
| | JSR | PRINT | 872 | 32 | 166 | 03 |
| | JMP | CLNUP | 875 | 76 | 122 | 03 |

```
ALPHA   LDA  CHAR     878   173  252   03
        CLC           881    24
        ADC    #40    882   105   64
        STA  PCHAR    884   141  255   03
        JSR  PRINT    887    32  166   03
CLNUP   INC  COUNT    890   238  251   03
        LDA  COUNT    893   171  251   03
        CMP    #28    896   201   40
        BEQ  NEWL     898   240   13
        INX           900   232
        TAX           901   138
        BNE  NEXTC    902   208   06
        INC    869    904   238   89   03
        INC    834    907   238   66   03
NEXTC   JMP  LOOP2    910    76   87   03
NEWL    LDA    #0     913   169   00
        STA  COUNT    915   141  251   03
        INX           918   232
        JMP  LOOP1    919    76   64   03
FINDR   LDA   #80     922   169  128
        STA    834    924   141   66   03
        STA    860    927   141   89   03
        RTS           930    96
PRINT   LDA   #FF     934   169  255
        STA  PADD     936   141   67  232
        LDA  PCHAR    939   173  255   03
        STA  CHAR     942   141  252   03
        STX   TMPX    945   142  253   03
        JSR   DELAY   948    32  230   03
        LDA   SAD     951   169   79  232
        AND    #FE    954    41  254
        STA   SAD     956   141   79  232
        JSR   DELAY   959    32  230   03
        LDX    #08    962   162   08
OUT1    LDA   SAD     964   173   79  232
        AND   #FE     967    41  254
        LSR   CHAR    969    78  252   03
        ADC   #00     972   105   00
        STA   SAD     974   141   79  232
        JSR   DELAY   977    32  230   03
        DEX           980   202
        BNE   OUT1    981   208  237
        LDA   SAD     983   173   79  232
        ORA   #01     986    09   01
        STA   SAD     988   141   79  232
        JSR   DELAY   991    32  230   03
        LDX   TMPX    994   174  253   03
        RTS           997    96
DELAY   LDA   #02     998   169   02
        STA   TIMH   1000   141  254   03
        LDA   #52    1003   169   82
DE2     SEC          1005    56
DE4     SBC   #01    1006   233   01
        BCS   DE3    1008   176   03
        DEC   TIMH   1010   206  254   03
DE3     LDY   TIMH   1013   172  254   03
        BPL   DE2    1016    16  243
        RTS          1018

COUNT   (1019)
CHAR    (1020)
TMPX    (1021)
TIMH    (1022)
PCHAR   (1023)
```

# SHAPING UP YOUR APPLE

Michael Faraday
246 Bronxville Road
Bronxville, NY 10708

Even though, as a programming novice, it took me a while to take on Apple II's Hi-Resolution Graphics I have to admit that the seeming complexity of constructing a Shape Table held a certain fascination for me from the first time I opened the Reference Manual. With Gary Dawkin's delightful program appearing in Creative Comput- ing

delightful program appearing in Creative Computing recently there is no longer any real need to apply the original technique, but a good understanding of something never hurt anyone, if only to verify other working arrangements.

If you have a TI Programmer, or any convenient way of converting from one base to another, here's a simplified method of untangling that unsightly jumble of arrows and binary digits on page 53 of the "Big Red Book". The key is in recognizing that the conversion chart is nothing more than an OCTal representation of our 8-bit

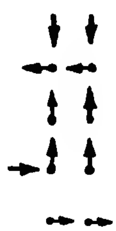
| A/B | C | OCT |
|---|---|---|
| 000 | 00 | 0 |
| 001 | 01 | 1 |
| 010 | 10 | 2 |
| 011 | 11 | 3 |
| 100 | | 4 |
| 101 | | 5 |
| 110 | | 6 |
| 111 | | 7 |

To the Code list we will add the OCTal number that each arrow represents.

byte. OCTal is binary broken into groups of three just as HEX is binary broken into groups of four. The fog lifts a little and we can now see why the "C" digit is limited to two bits: we only have a total of eight to start with. Looking a little further along the same page we come to the Conversion Codes and it's here we can begin to make things really easy.

| C | B | A |
|---|---|---|
| 0 0 | 0 1 0 | 0 1 0 |
| 0 0 | 1 1 1 | 1 1 1 |
| 0 0 | 1 0 0 | 0 0 0 |
| 0 1 | 1 0 0 | 1 0 0 |
| 0 0 | 1 0 1 | 1 0 1 |
| ... | ... | ... |

C B A

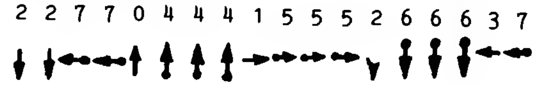


To the Code list we will add the OCTal number each arrow represents.

Going back to the original example in the manual we can replace the entire chart of binary digits with an OCTal number put directly above our "unwrapped" arrows, like so:

OCT 2 2 7 7 0 4 4 4 1 5 5 5 2 6 6 6 3 7

Shape



We are going to construct either two- or three-digit numbers from this list and now come the only rules required to deal with in the whole procedure:

1. While always trying to make a three-digit number, the "last" digit of a three-digit group can ONLY be a 1, 2 or 3 (remember that the "C" digit is only 2 binary digits, which can represent the OCTal number three at most).

2. As usual, these numbers appear Least Significant Digit first and therefore the "last" digit is, in reality, the first digit of the new OCTal number.

So we can now divide the long string of numbers into two- and three-digit, reverse-order OCTal numbers with slashes:

OCTal 2 2/7 7/0 4/4 4 1/5 5/5 2/6 6/6 3/7

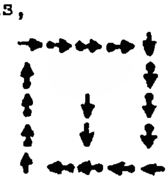
"unwrap" this list, reversing digits as we go:

"unwrap" this list, reversing digits as we go, and converting to HEX:

| OCT | HEX |
|---|---|
| 22 | 12 |
| 77 | 3F |
| 40 | 20 |
| 144 | 64 |
| ... | ... |

Even this can be a bit tedious and since I find the arrow Code conversion very easy to remember - No Plot, Up Clockwise to Left = 0 to 3; Plot, Up Clockwise to Left = 4 to 7 - I draw my diagrams on graph paper using these OCTal numbers only.

Thus, becomes



```
1 5 5 5 2
4       6
4   2   6
4   2   6
0 7 7 7 3
```

Some caveats. It's still a good idea to draft an original diagram with plain dots just to get the shape and scale to your liking. This also becomes a handy guide for the debugging you're almost certain to have to do. And too, it makes great fun for your non-computer friends who might like to play Connect-the-Dots after a couple of beers.

A big problem keeps cropping up using the scale feature. It seems that when blowing up the original drawing the Apple II uses the direction of motion associated with the plotted points as a base reference for the additional points. This often leads to strangely assymetrical pictures in larger scale with "lines" of dots going in unexpected directions. As always, a little playing around can really make you feel good. Have fun.

Hexidecimal - Octal Conversion Table

| HEX | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 1 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 |
| 2 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 |
| 3 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 |
| 4 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 |
| 5 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 | 130 | 131 | 132 | 133 | 134 | 135 | 136 | 137 |
| 6 | 140 | 141 | 142 | 143 | 144 | 145 | 146 | 147 | 150 | 151 | 152 | 153 | 154 | 155 | 156 | 157 |
| 7 | 160 | 161 | 162 | 163 | 164 | 165 | 166 | 167 | 170 | 171 | 172 | 173 | 174 | 175 | 176 | 177 |
| 8 | 200 | 201 | 202 | 203 | 204 | 205 | 206 | 207 | 210 | 211 | 212 | 213 | 214 | 215 | 216 | 217 |
| 9 | 220 | 221 | 222 | 223 | 224 | 225 | 226 | 227 | 230 | 231 | 232 | 233 | 234 | 235 | 236 | 237 |
| A | 240 | 241 | 242 | 243 | 244 | 245 | 246 | 247 | 250 | 251 | 252 | 253 | 254 | 255 | 256 | 257 |
| B | 260 | 261 | 262 | 263 | 264 | 265 | 266 | 267 | 270 | 271 | 272 | 273 | 274 | 275 | 276 | 277 |
| C | 300 | 301 | 302 | 303 | 304 | 305 | 306 | 307 | 310 | 311 | 312 | 313 | 314 | 315 | 316 | 317 |
| D | 320 | 321 | 322 | 323 | 324 | 325 | 326 | 327 | 330 | 331 | 332 | 333 | 334 | 335 | 336 | 337 |
| E | 340 | 341 | 342 | 343 | 344 | 345 | 346 | 347 | 350 | 351 | 352 | 353 | 354 | 355 | 356 | 357 |
| F | 360 | 361 | 362 | 363 | 364 | 365 | 366 | 367 | 370 | 371 | 372 | 373 | 374 | 375 | 376 | 377 |

## APPLE II STARWARS THEME


Andrew H. Eliason
28 Charles Lane
Falmouth, MA 02540


Just for the fun of it, here are some routines to create something which sounds like the main battle scene from STARWARS. Enjoy!

Apple II Startrek Sounds Routine
Dis-assembler Listing

```
*3FA1L

3FA1-   A0 0E      LDY   #$0E
3FA3-   A2 00      LDX   #$00
3FA5-   8A         TXA
3FA6-   18         CLC
3FA7-   E9 01      SBC   #$01
3FA9-   D0 FC      BNE   $3FA7
3FAB-   8D 30 C0   STA   $C030
3FAE-   E8         INX
3FAF-   E0 8C      CPX   #$8C
3FB1-   D0 F2      BNE   $3FA5
3FB3-   88         DEY
3FB4-   D0 ED      BNE   $3FA3
3FB6-   60         RTS
3FB7-   00         BRK
3FB8-   00         BRK
3FB9-   00         BRK
3FBA-   00         BRK
3FBB-   00         BRK
3FBC-   00         BRK
3FBD-   00         BRK
*
```

Load via monitor starting at 3FA1:

```
3FA1.3FB6

3FA1- A0 0E A2 00 8A 18 E9
3FA8- 01 D0 FC 8D 30 C0 E8 E0
3FB0- 8C D0 F2 88 D0 ED 60
*
```

Enter BASIC and set HIMEM:16288.
Enter this program and RUN:

```
LIST

>LIST
  10 PRINT "STAR BATTLE SOUND EFFECTS"
  20 I= RND (15)+1: REM    SHOTS
  30 J= RND (11)*10+120: REM  DURATION
  40 POKE 16290,I: POKE 16304,J
  50 CALL 16289
  60 N= RND (1000): FOR K=1 TO N: NEXT K
  70 GOTO 20
 999 END
```

Try I = RND(30)+1 and J = RND(255).

The above material is based on the "Phaser" sound effect from Apple II Startrek.

# APPLE PI

Robert J. Bishop
1143 W. Badillo, Apt E
Covina, CA 91722

Everyone knows that the value of Pi is about 3.1416. In fact, its value was known this accurately as far back as 150 A.D. But it wasn't until the sixteenth century that Francisco Vieta succeeded in calculating Pi to ten decimal places.

Around the end of the sixteenth century the German mathematician, Ludolph von Ceulen, worked on calculating the value of Pi until he died at the age of 70. His efforts produced Pi to 35 decimal places.

During the next several centuries a great deal of effort was spent in computing the value of PI to evern greater precision. In 1699 Abraham Sharp calculated Pi to 71 decimal places. By the mid 1800's its value was known to several hundred decimal places. Finally, in 1873, an English mathematician, Shanks, determined Pi to 707 decimal places, an accuracy which remained unchallenged for many years.

I was recently rereading my old copy of Kasner & Newman"s Mathematics and the Imagination

I was recently rereading my old copy of Kasner & Newman's Mathematics and Imagination (Simon & Schuster, 1940), where I found the series expansion:

$$\pi = \sum_{k=1}^{\infty} \frac{16(-1)^{k+1}}{(2k-1)5^{2k-1}} - \sum_{k=1}^{\infty} \frac{4(-1)^{k+1}}{(2k-1)239^{2k-1}}$$

The book indicated that this series converged rather quickly but "... it would require ten years of calculation to determine Pi to 1000 decimal places." Clearly this statement was made before modern digital computers were available. Since then, Pi has been computed to many thousands of decimal places. But Kasner & Newman's conjecture of a ten-year calculation for Pi aroused my curiousity to see just how long it would take my little Apple-II computer to perform the task.

## Program Description

My program to compute the value of Pi is shown in Figure 1. It was written using the Apple II computer's Integer BASIC and requires a 16K system (2K for the program inself; 12K for data storage). The program is fairly straightforward but a brief discussion may be helpful.

The main calculation loop consists of lines 100 through 300; the results are printed in lines 400 through 600. The second half of the listing contains the multiple precision arithmetic subroutines. The division, addition, and subtraction routines start at lines 1000, 2000, and 3000, respectively.

In order to use memory more efficiently, PEEK and POKE statements were used for arrays instead of DIM statements. Three such arrays are used by the program: POWER, TERM, and RESULT. Each are up to 4K bytes long and start at the memory locations specified in line 50 of the program.

The three arrays mentioned above each store partial and intermediate results of the calculations. Each byte of an array contains either one or two digits, depending on the value of the variable, TEN. If the number of requested digits for Pi is less than about 200, it is possible to store two digits per byte; otherwise, each byte must contain no more than one digit. (The reason for this distinction occurs in line 1070 where an arithmetic overflow can occur when trying to evaluate higher order terms of the series if too many digits are packed into each byte.)

The program evaluates the series expansion for Pi until the next term of the series results in a value less than the requested precision. Line 1055 computes the variable, ZERO, which can be tested to see if an underflow in precision has occurred. This value is then passed back to the main program where, in line 270, it determines whether or not the next term of the series is needed.

## Results

Figure 2 shows the calculated value of Pi to 1000 decimal places. Running the program to get these results took longer than it did to write the program! (The program ran for almost 40 hours before it spit out the answer.) However it took less than two minutes to produce Pi to 35 decimal places, the same accuracy to which Ludolph von Ceulen spent his whole life striving for!

Since the program is written entirely in BASIC it is understandably slow. By rewriting all or part of it in machine language its performance could be vastly improved. However, I will leave this implementation as an exercise for anyone who is interested in pursuing it.

Figure 1.

Program Listing

```
>LIST
    0 REM  *** APPLE-PI ***
      WRITTEN BY: BOB BISHOP
    5 CALL -936: VTAB 10: TAB 5: PRINT
      "HOW MANY DIGITS DO YOU WANT"
      ;
   10 INPUT SIZE
   15 CALL -936
   20 TEN=10: IF SIZE>200 THEN 50

   30 TEN=100: SIZE=(SIZE+1)/2
   50 POWER=4096: TERM=8192: RESULT=
      12288
   60 DIV=1000: ADD=2000: SUB=3000:
      INIT=4000: COPY=5000
   70 DIM CONSTANT(2): CONSTANT(1)
      =25: CONSTANT(2)=239
```

```
100 REM  MAIN LOOP
125 FOR PASS=1 TO 2
150 GOSUB INIT
200 GOSUB COPY
210 POINT=TERM: DIVIDE=EXP:  GOSUB
    DIV
220 IF SIGN>0 THEN GOSUB ADD
230 IF SIGN<0 THEN GOSUB SUB
240 EXP=EXP+2: SIGN=-SIGN
250 POINT=POWER: DIVIDE=CONSTANT(
    PASS):  GOSUB DIV
260 IF PASS=2 THEN GOSUB DIV
270 IF ZERO<>0 THEN 200
300 NEXT PASS
400 REM  PRINT THE RESULT
500 PRINT : PRINT
510 PRINT "THE VALUE OF PI TO "
    ;(TEN/100+1)*SIZE;" DECIMAL PLAC
    ES: ": PRINT
520 PRINT PEEK (RESULT);". ";
530 FOR PLACE=RESULT+1 TO RESULT+
    SIZE
540 IF TEN=10 THEN 570
560 IF PEEK (PLACE)<10 THEN PRINT
    "0";
570 PRINT PEEK (PLACE);
580 NEXT PLACE
590 PRINT
600 END
1000 REM  DIVISION SUBROUTINE
1010 DIGIT=0: ZERO=0
1020 FOR PLACE=POINT TO POINT+SIZE
1030 DIGIT=DIGIT+ PEEK (PLACE)
1040 QUOTIENT=DIGIT/DIVIDE
1050 RESIDUE=DIGIT MOD DIVIDE
1055 ZERO=ZERO OR (QUOTIENT+RESIDUE)

1060 POKE PLACE, QUOTIENT
1070 DIGIT=TEN*RESIDUE
1080 NEXT PLACE
1090 RETURN
2000 REM  ADDITION SUBROUTINE
2010 CARRY=0
2020 FOR PLACE=SIZE TO 0 STEP -1

2030 SUM= PEEK (RESULT+PLACE)+ PEEK
     (TERM+PLACE)+CARRY
2040 CARRY=0
2050 IF SUM<TEN THEN 2080
2060 SUM=SUM-TEN
2070 CARRY=1
2080 POKE RESULT+PLACE, SUM
2090 NEXT PLACE
2100 RETURN
3000 REM  SUBTRACTION SUBROUTINE
3010 LOAN=0
3020 FOR PLACE=SIZE TO 0 STEP -1
3030 DIFFERENCE= PEEK (RESULT+PLACE)
     - PEEK (TERM+PLACE)-LOAN
3040 LOAN=0
3050 IF DIFFERENCE>=0 THEN 3080
3060 DIFFERENCE=DIFFERENCE+TEN
3070 LOAN=1
3080 POKE RESULT+PLACE, DIFFERENCE
3090 NEXT PLACE
3100 RETURN
4000 REM   INITIALIZE REGISTERS
4010 FOR PLACE=0 TO SIZE
4020 POKE POWER+PLACE, 0
4030 POKE TERM+PLACE, 0
4040 IF PASS=1 THEN POKE RESULT+
     PLACE, 0
4050 NEXT PLACE
4060 POKE POWER, 16/PASS ↑ 2
4070 IF PASS=1 THEN DIVIDE=5
4080 IF PASS=2 THEN DIVIDE=239
4090 POINT=POWER:  GOSUB DIV
4100 EXP=1: SIGN=3-2*PASS
4110 RETURN
5000 REM  COPY "POWER" INTO "TERM"
5010 FOR PLACE=0 TO SIZE
5020 POKE TERM+PLACE, PEEK (POWER+
     PLACE)
5030 NEXT PLACE
5040 RETURN
```

```
THE VALUE OF PI TO 1000 DECIMAL PLACES:

3. 14159265358979323846264338327950288419
7169399375105820974944592307816406286208
9986280348253421170679821480865132823066
4709384460955058223172535940812848111745
0284102701938521105559644622948954930381
9644288109756659334461284756482337867831
6527120190914564856692346034861045432664
8213393607260249141273724587006606315588
1748815209209628292540917153643678925903
6001133053054882046652138414695194151160
9433057270365759591953092186117381932611
7931051185480744623799627495673518857527
2489122793818301194912983367336244065664
3086021394946395224737190702179860943702
7705392171762931767523846748184676694051
3200056812714526356082778577134275778960
9173637178721468440901224953430146549585
3710507922796892589235420199561121290219
6086403441815981362977477130996051870721
1349999998372978049951059731732816096318
5950244594553469083026425223082533446850
3526193118817101000313783875288658753320
8381420617177669147303598253490428755468
7311595628638823537875937519577818577805
3217122680661300192787661119590921642019
96
```

Figure 2.

Pi to 1000 Decimal Places

# A SIMPLE 6502 ASSEMBLER FOR THE PET

Michael J. McCann
28 Ravenswood Terrace
Cheektowaga, NY 14225

Most computer hobbyists do all or most of their programming in BASIC. This is unfortunate since there is much to be gained from machine code level programming. On the average, machine language programs are 100 times faster than their BASIC equivalents. In addition, machine language programs are very compact, making efficient use of memory. I have written a simple 6502 assembler in Commodore BASIC (see listing) with the following functions:

1. Input source code and assemble
2. Save object code on tape
3. Load object code from tape
4. Run machine language program with SYS
5. Run machine language program with USR
6. List machine language program

INPUT SOURCE CODE AND ASSEMBLE
-Symbolic addresses and operands are not permitted
-All addresses and operands must be supplied in base 10
-Each line of source code is assembled after entry
-Source code is inputted in the following format:
 (mnemonic)(one or more spaces)(operand)
-Three pseudoinstructions are supported
 ORG-Start with this address
 NOTE: if the user does not specify the origin, it will be set at 826 base 10
 DC-Define constant, place the operand value in the next location in memory
 END-End of program source code

SAVE OBJECT CODE ON TAPE
-Object code saved under file name supplied by user
-Origin address saved with program

LOAD OBJECT CODE FROM TAPE
-Loads object program under file name supplied by user
-Object code is stored in memory with the same origin address used when the program was assembled

RUN MACHINE LANGUAGE PROGRAM WITH SYS
-Transfers control of the 6502 to an address supplied by the user

RUN MACHINE LANGUAGE PROGRAM WITH USR
-Transfers a user supplied value to the 6502 accumulator
-Transfers control of the 6502 to an address supplied by the user

LIST MACHINE LANGUAGE PROGRAM
-Listing is produced by disassembling object code
-Disassembly is in the following format:
 (decimal address)(hexadecimal address)(byte#1)(byte#2)(byte#3)(mnemonic)(operand)

The following areas of memory are available for your machine language programs when this assembler is in memory: locations 7884-8184 and, if tape #2 is not used, locations 826-1024.

There are two ways of returning control to BASIC from machine language. The RTS (Return from Subroutine) instruction may be used at any time except when in a user machine language subroutine. RTS returns control to the calling BASIC program. In contrast the BRK (Force Break) instruction does not return control to the calling BASIC program; instead control is returned to the user, i.e. system prints READY with the cursor.

I have included a short machine language program. When run this program will leave a pattern of small white dots on the upper half of PET's CRT.

SAMPLE MACHINE LANGUAGE PROGRAM LISTING

```
826 033A  A9 66     LDAIM   102
828 033C  A2 00     LDXIM   0
830 033E  9D 00 80  STAX    32768
833 0341  E8        INX
834 0342  F0 03     BEQ     3
836 0344  4C 3E 03  JMP     830
839 0347  EA        NOP
840 0348  EA        NOP
841 0349  9D 00 81  STAX    33024
844 034C  E8        INX
845 034D  F0 03     BEQ     3
847 034F  4C 49 03  JMP     841
850 0352  00        BRK
```

SAMPLE MACHINE LANGUAGE PROGRAM
AS INPUTTED FROM THE KEYBOARD

```
? ORG 826
? LDAIM 102
? LDXIM 0
? STAX 32768
? INX
? BEQ 3
? JMP 830
? NOP
? NOP
? STAX 33024
? INX
? BEQ 3
? JMP 841
? BRK
? END
```

```
1   REM 6502 ASSEMBLER PROGRAM
2   REM BY MICHAEL J. MCCANN
3   REM FOR USE ON THE COMMODORE PET
10  DIM MN$(256),BY%(256),CO$(16)
20  FOR E=0 TO 255
30  READ MN$(E),BY%(E)
40  NEXT
60  FOR E=0 TO 15
70  READ CO$(E)
80  NEXT
90  PRINT CHR$(147):PRINT
100  PRINT"1-INPUT SOURCE CODE AND ASSEMBLE":PRINT
110  PRINT"2-SAVE OBJECT CODE ON TAPE":PRINT
120  PRINT"3-LOAD OBJECT CODE FROM TAPE":PRINT
130  PRINT"4-RUN MACHINE LANGUAGE PROGRAM WITH SYS"
140  PRINT"5-RUN MACHINE LANGUAGE PROGRAM WITH USR"
150  PRINT"6-LIST MACHINE LANGUAGE PROGRAM"
180  GET A$:IF A$="" GOTO 180
190  IF VAL(A$)=0 OR VAL(A$)>6 GOTO 180
200  ON VAL(A$) GOSUB 14000,20000,9000,10000,11000,2900
210  GOTO 90
1000  SX=INT(DC/16)
1010  UN=DC-(SX*16)
1020  SX$=CO$(SX)
1030  UN$=CO$(UN)
1040  HX$+SX$+UN$
1050  RETURN
2900  PRINT CHR$(147)
2910  INPUT"START ADDRESS";AD:I=0
3000  IF I=24 GOTO 5050
3001  I=I+1
3005  IB=PEEK(AD)
3015  IF MN$(IB)<>"NULL" GOTO 3050
3025  DC=IB:GOSUB 1000:GOSUB 13000
3030  PRINT AD;AD$ TAB(12) HX$ "*"
3040  AD=AD+1:GOTO 3000
3050  ON BY%(IB) GOTO 3060,3090,4050
3060  DC=IB:GOSUB 1000:GOSUB 13000
3070  PRINT AD;AD$ TAB(12);HX$;TAB(21);MN$(IB)
3075  AD=AD+1
3080  GOTO 5030
3090  DC=IB:GOSUB 1000
4000  B1$=HX$
4010  DC=PEEK(AD+1):GOSUB 1000
4011  B2$=HX$
4024  GOSUB 13000:P=DC
4030  PRINT AD;AD$ TAB(12);B1$;" ";B2$;TAB(21);MN$(1B);TAB(27);P
4035  AD=AD+2
4040  GOTO 5030
4050  DC=IB:GOSUB 1000
4060  B1$=HX$
4070  DC=PEEK(AD+1):GOSUB 1000
4080  B2$=HX$
4090  DC=PEEK(AD+2):GOSUB 1000
```

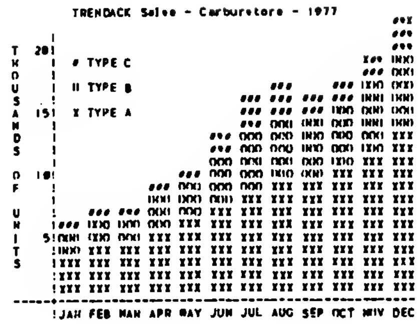
```
5000  B3$=HX$
5010  OP=PEEK(AD+1)+(PEEK(AD+2)*256)
5011  GOSUB 13000
5020  PRINT AD;AD$ TAB(12);B1$;" ";B2$;" ";B3$;TAB(21);MN$(IB);TAB(27) OP
5025  AD=AD+3
5030  GOTO 3000
5050  GET A$:IF A$="" GOTO 5050
5051  IF A$=CHR$(19) THEN I=0:RETURN
5052  IF A$<>CHR$(13) GOTO 5050
5070  I=0:PRINT CHR$(147)
5080  GOTO 3000
6000  DATA BRK,1,ORAIX,2,NULL,0,NULL,0,NULL,0,ORAZ,2,ASL,2,NULL,0,PHP,1
6010  DATA ORAIM,2,ASLA,1,NULL,0,NULL,0,ORA,3,ASL,3,NULL,0,BPL,2,ORAIY,2
6020  DATA NULL,0,NULL,0,NULL,0,ORAZX,2,ASLZX,2,NULL,0,CLC,1,ORAY,3
6030  DATA NULL,0,NULL,0,NULL,0,ORAX,3,ASLX,3,NULL,0,JSR,3,ANDIX,2,NULL,0
6040  DATA NULL,0,BITZ,2,ANDZ,2,ROLZ,2,NULL,0,PLP,1,ANDIM,2,ROLA,1,NULL,0
6050  DATA BIT,3,AND,3,ROL,3,NULL,0,BMI,2,ANDIY,2,NULL,0,NULL,0,NULL,0
6060  DATA ANDZX,2,ROLZX,2,NULL,0,SEC,1,ANDY,3,NULL,0,NULL,0,ANDX,3
6070  DATA ROLX,3,NULL,0,RTI,1,EORIX,2,NULL,0,NULL,0,NULL,0,EORZ,2,LSRZ,2
6080  DATA NULL,0,PHA,1,EORIM,2,LSRA,1,NULL,0,JMP,3,EOR,3,LSR,3,NULL,0
6090  DATA BVC,2,EORIY,2,NULL,0,NULL,0,NULL,0,EORZX,2,LSRZX,2,NULL,0
6100  DATA CLC,1,EORY,3,NULL,0,NULL,0,NULL,0,EORX,3,LSRX,3,NULL,0,RTS,1
6110  DATA ADCIX,2,NULL,0,NULL,0,NULL,0,ADCZ,2,RORZ,2,NULL,0,PLA,1,ADCIM,2
6120  DATA RORA,1,NULL,0,JMI,3,ADC,3,ROR,3,NULL,0,BVS,2,ADCIY,2,NULL,0
6130  DATA NULL,0,NULL,0,ADCZX,2,RORZX,2,NULL,0,SEI,1,ADCY,3,NULL,0,NULL,0
6140  DATA NULL,0,ADCX,3,RORX,3,NULL,0,NULL,0,STAIX,2,NULL,0,NULL,0,STYZ,2
6150  DATA STAZ,2,STXZ,2,NULL,0,DEY,1,NULL,0,TXA,1,NULL,0,STY,3,STA,3
6160  DATA STX,3,NULL,0,BCC,2,STAIY,2,NULL,0,NULL,0,STYZX,2,STAZX,2,STXZY,2
6170  DATA NULL,0,TYA,1,STAY,3,TXS,1,NULL,0,NULL,0,STAX,3,NULL,0,NULL,0
6180  DATA LDYIM,2,LDAIX,2,LDXIM,2,NULL,0,LDYZ,2,LDAZ,2,LDXZ,2,NULL,0
6190  DATA TAY,1,LDAIM,2,TAX,1,NULL,0,LDY,3,LDA,3,LDX,3,NULL,0,BCS,2
6200  DATA LDAIY,2,NULL,0,NULL,0,LDYZX,2,LDAZX,2,LDXZY,2,NULL,0,CLV,1
6210  DATA LDAY,3,TSX,1,NULL,0,LDYX,3,LDAX,3,LDXY,3,NULL,0,CPYIM,2,CMPIX,2
6220  DATA NULL,0,NULL,0,CPYZ,2,CMPZ,2,DECZ,2,NULL,0,INY,1,CMPIM,2,DEX,1
6230  DATA NULL,0,CPY,3,CMP,3,DEC,3,NULL,0,BNE,2,CMPIY,2,NULL,0,NULL,0
6240  DATA NULL,0,CMPZX,2,DECZX,2,NULL,0,CLD,1,CMPY,3,NULL,0,NULL,0,NULL,0
6250  DATA CMPX,3,DECX,3,NULL,0,CPXIM,2,SBCIX,2,NULL,0,NULL,0,CPX,2,SBCZ,2
6260  DATA INCZ,2,NULL,0,INX,1,SBCIM,2,NOP,1,NULL,0,CPX,3,SBC,3,INC,3
6270  DATA NULL,0,BEQ,2,SBCIY,2,NULL,0,NULL,0,NULL,0,SBCZX,2,INCZX,2,NULL,0,SED,1
6280  DATA SBCY,3,NULL,0,NULL,0,NULL,0,SBCX,3,INCX,3,NULL,0
6290  DATA 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
9000  PRINT CHR$(147)
9010  INPUT "ENTER FILE NAME";N$
9020  OPEN 1,1,0,N$
9030  INPUT#1,ZZ
9040  INPUT#1,EN
9050  FOR AD=ZZ TO EN
9060  INPUT#1,DA$
9070  POKE AD,DA$
9080  NEXT
9090  CLOSE 1
9100  RETURN
```

```
10000  PRINT CHR$(147)
10010  INPUT "ENTER ADDRESS IN BASE 10";AD
10015  IF AD>65535 GOTO 10000
10020  SYS(AD)
10030  RETURN
11000  PRINT CHR$(147)
11010  INPUT"ENTER ACCUMULATOR VALUE";AC
11015  IF AC<0 OR AC>255 GOTO 11010
11020  INPUT"ENTER ADDRESS IN BASE 10";AD
11030  POKE 2,INT(AD/256)
11040  POKE 1,AD-(INT(AD/256)*256)
11050  X=USR(AC)
11060  RETURN
13000  A=AD:S3=INT(AD/4096)
13002  A=A-S3*4096
13010  S2=INT(A/256)
13012  A=A-S2*256
13020  S=INT(A/16)
13060  U=AD-(S3*4096+S2*256+S*16)
13070  S3$=CO$(S3)
13080  S2$=CO$(S2)
13090  S$=CO$(S)
13100  U$=CO$(U)
13110  AD$=S3$+S2$+S$+U$
13120  RETURN
14000  PRINT CHR$(147):AD=826:ZZ=826
14010  PRINT "(MNEMONIC)(SPACE)(OPERAND)"
14020  GOSUB 15000
14030  F=0
14040  FOR E=0 TO 255
14050  IF MN$=MN$(E) THEN BY=BY%(E):F=1:CD=E:E=256
14060  NEXT
14070  IF F=0 GOTO 14260
14080  ON BY GOSUB 14100,14130,14180
14090  GOTO 14020
14100  POKE AD,CD
14110  AD=AD+1
14120  RETURN
14130  IF OP>255 OR OP<0 THEN PRINT "ERROR":RETURN
14140  POKE AD,CD
14150  POKE AD+1,OP
14160  AD=AD+2
14170  RETURN
14180  IF OP>65535 OR OP<0 THEN PRINT "ERROR":RETURN
14190  POKE AD,CD
14200  B2=INT(OP/256)
14210  B1=OP-(B2*256)
14220  POKE AD+1,B1
14230  POKE AD+2,B2
14240  AD=AD+3
14250  RETURN
14260  IF MN$="ORG" OR MN$="END" OR MN$="DC" GOTO 14280
14270  PRINT "ERROR":GOTO 14020
14280  IF MN$="ORG" GOTO 14300
14290  GOTO 14340
14300  IF FO=1 THEN PRINT "ERROR":GOTO 14020
14310  FO=1
14320  AD=OP:ZZ=OP
14330  GOTO 14020
```

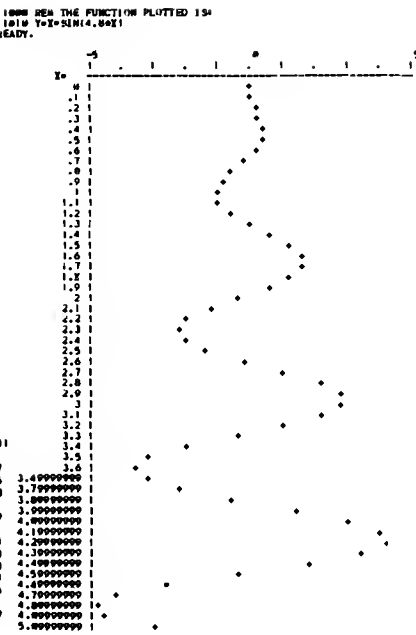
```
14340  IF MN$="END" GOTO 14360
14350  GOTO 14380
14360  EN=AD-1
14370  RETURN
14480  POKE AD,OP
14510  AD=AD+1
14520  GOTO 14020
15000  INPUT A$
15010  IF LEN(A$)<3 THEN PRINT "ERROR":GOTO 15000
15020  IF LEN(A$)=3 THEN MN$ A$:OP=0:RETURN
15030  S=0:FOR M=1 TO LEN(A$)
15040  IF MID$(A$,M,1)=" " THEN S=M:M=LEN(A$)
15050  NEXT
15060  IF S=0 THEN MN$=A$:RETURN
15070  MN$=LEFT$(A$,S-1)
15080  OP=VAL(RIGHT$(A$,LEN(A$)-S))
15090  RETURN
20000  PRINT CHR$(147):SZ=0
20010  INPUT "ENTER PROGRAM NAME";N$
20020  OPEN 1,1,1,N$
20030  PRINT#1,ZZ:DA%=ZZ:GOSUB 20110
20040  PRINT#1,EN:DA%=EN:GOSUB 20110
20050  FOR AD=ZZ TO EN
20060  DA%=PEEK(AD)
20070  PRINT#1,DA%:GOSUB 20110
20080  NEXT
20090  CLOSE 1
20100  RETURN
20110  SZ=LEN(STR$(DA%))+SZ+1
20120  IF SZ<192 THEN RETURN
20130  POKE 59411,53
20140  T=TI
20150  IF (TI-T)<6 GOTO 20150
20160  POKE 59411,61
20170  SZ=SZ-191
20180  RETURN
```

# COMMODORE PET HARD COPY OUTPUT USING PET ADA 1200

```
1 OPEN 6,8:CMD 6:LIST
10 REM ARCSIN AND ARCCOS FUNCTIONS FOR THE COMMODORE PET
70 REM "Listed on a GE TermiNet 300
80 REM "using a CmC ADA 1200.
90 REM
100 REM OPEN OUTPUT FILE ON DEVICE #8.
110 OPEN 5,8
120 REM
500 REM GET A SINE VALUE
510 INPUT S
520 C=S
530 REM
1000 REM THE SINE OF THE ANGLE IS S
1010 REM IF THE SIN IS IN THE RANGE OF -1 TO 1, THEN COMPUTE.
1020 IF S<1 AND S>-1 GOTO 1050
1030 AS=90:GOTO2000
1040 REM THE ARCSINE IS AS
1050 AS=ATN(S/((1-S*S)^.5))
1060 REM THE RESULT IS IN RADIANS. CONVERT TO DEGREES.
1070 AS=AS*180/π
2000 REM THE COSINE OF THE ANGLE IS C
2010 REM IF THE COSINE IS IN THE RANGE OF -1 TO 1,
2011 REM       AND NOT = TO 0, THEN COMPUTE
2020 IF C<>0 THEN 2040
2030 AC=90:GOTO 3000
2040 IF C<1 AND C>-1 THEN 2070
2050 AC=0:GOTO 3000
2060 REM THE ARCCOS IS AC
2070 AC=ATN(((1-C*C)^.5/C))
2080 REM THE RESULT IS IN RADIANS. CONVERT TO DEGREES.
2090 AC=AC*180/π
3000 PRINT#5," SIN,COS       ARCSIN        ARCCOS"
3010 PRINT#5,S;
3020 FOR N=1 TO 13-LEN(STR$(S)):PRINT#5," ";:NEXT
3030 PRINT#5,AS;
3040 FOR N=1 TO 13-LEN(STR$(AS)):PRINT#5," ";:NEXT
3050 PRINT#5,AC
3100 GOTO 500
READY.
```




| X | X^2 | X^3 | X^PI | X^5.25 | X^.5 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 50 | 2500 | 125000 | 217506.221 | 830983735 | 7.07106781 |
| 100 | 10000 | 1000000 | 1919487.50 | 3.16227767E+10 | 10 |
| 150 | 22500 | 3375000 | 6861077.20 | 2.65753572E+11 | 12.2474487 |
| 200 | 40000 | 8000000 | 16939435.3 | 1.20339209E+12 | 14.1421356 |
| 250 | 62500 | 15625000 | 34146855.2 | 3.88315787E+12 | 15.8113883 |
| 300 | 90000 | 27000000 | 60540854.0 | 1.01131532E+13 | 17.3205081 |
| 350 | 122500 | 42875000 | 98271022.0 | 2.27173273E+13 | 18.7082869 |
| 400 | 160000 | 64000000 | 149498141 | 4.57946723E+13 | 20 |
| 450 | 202500 | 91125000 | 216427747 | 8.49895829E+13 | 21.2132034 |
| 500 | 250000 | 125000000 | 301345240 | 1.47772127E+14 | 22.3606798 |
| 550 | 302500 | 166375000 | 406540006 | 2.43727268E+14 | 23.4520788 |
| 600 | 360000 | 216000000 | 534342301 | 3.84052201E+14 | 24.4948974 |
| 650 | 422500 | 274625000 | 687112634 | 5.85861722E+14 | 25.4950976 |
| 700 | 490000 | 343000000 | 867219597 | 8.64499043E+14 | 26.4575131 |
| 750 | 562500 | 421875000 | 1.07713790E+09 | 1.24185712E+15 | 27.3861279 |




## RS-232 PRINTER ADAPTER FOR THE COMMODORE PET

The CONNECTICUT microCOMPUTER ADApter model 1200 is the first in a line of peripheral adapters for the COMMODORE PET. The CmC ADA 1200 drives an RS-232 printer from the PET IEEE-488 bus. The CmC ADA 1200 allows the PET owner to obtain hard copy program listings, and to type letters, manuscripts, mailing labels, tables of date, pictures, invoices, graphs, checks, needlepoint patterns, etc., using a standard RS-232 printer.

The CmC ADA model 1200B comes assembled and tested, without power supplies, case, or RS-232 connector for $98.50. The CmC ADA 1200C comes complete for $169.00. Specify baud rate when ordering. (300 baud is supplied unless otherwise requested. Instructions for changing the baud rate are included.)

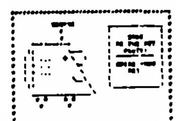
## WORD PROCESSOR FOR THE COMMODORE PET

CONNECTICUT microCOMPUTER now has a word processor program for the COMMODORE PET. This program permits composing and printing letters, flyers, advertisements, manuscripts, articles, etc., using the COMMODORE PET and an RS-232 printer.

Script directives include line length, left margin, centering, and skip. Edit commands allow the user to insert lines, delete lines, move lines, change strings, save onto cassette, load from cassette, move up, move down, print and type.

The CmC Word Processor Program addresses an RS-232 printer through a CmC printer adapter.

The CmC Word Processor Program is available for $29.50.

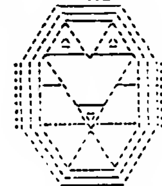


## RS-232 TO CURRENT LOOP/TTL ADAPTER

The CmC ADApter model 400 has two circuits. The first converts an RS-232 signal to a 20 ma current loop signal, and the second converts a 20 ma current loop signal to an RS-232 signal. With this device a computer's teletype port can be used to drive an RS-232 terminal, or vice versa, without modification of the port. The CmC ADA 400 can also be paralelled to drive a teletype or RS-232 printer while still using the computer's regular terminal. The CmC ADA 400 can easily be modified to become an RS-232 to TTL and TTL to RS-232 ADApter. The CmC ADA 400 does not alter the baud rate and uses standard power supplies. The current loop is isolated from the RS-232 signal by optoisolators.

The CmC ADA 400 is the perfect partner for KIM if you want to use an RS-232 terminal instead of a current loop teletype.

The CmC ADA 400S comes with drilled, plated through solder pads and sells for $24.50. The CmC ADA 400B comes with barrier strips and screw terminals and sells for $29.50.



This announcement was composed on a COMMODORE PET and printed on a GE TermiNet using a CmC ADA 1200C printer adapter and the CmC Word Processor Program.

Mike Rowe
P.O. Box 3
S. Chelmsford, MA 01824

Name: LABELER
System: TIM based or any 6502 based system
Memory: 1K
Language: Assembly
Hardware: Paper Tape Punch on TTY
Description: This program punches legible characters on a paper tape and is useful for the labeling of punched paper tapes. A 64 character sub-set of ASCII is used. There is limited editing capability on the data. There are a number of options for character size, starting address and TIM or I/O independent code.
Copies: Not Specified
Price: $4.00
Includes: Commented source listing, operating and modifying instructions, and a hex tape.
Ordering Info: Specify the following:
 Char Size 5x5 or 5x8
 Starting address 0200 or 1000
 System TIM or I/O Independent
Author: Gil House
Available from:
 Gil House
 P.O. Box 158
 Clarksburg, MD 20734

Name: HUEY
System: Any 6502 based system.
Memory: 2.5K
Language: Assembly
Hardware: ASCll I/O device.
Description: HUEY-65 is a scientific calculator program for the 6502 microprocessors. It operates from your ASCIl keyboard like a calculator; will output through your routines to a TV screen or Teletype; is preprogrammed to do trig functions, natural and common logs, exponential functions and other goodies; and is programmable for many other functions (financial, accounting, mathematics, engineering, etc.) you would like to call at the press of a single key.
Copies: Not Specified.
Price: Hex Dump at any even page - $5.00
 Manual and Listings - $20.00
Ordering Info: Specify starting address.
Author: Don Rindsberg
Available from:
 The BIT Stop
 P.O. Box 973
 Mobile, AL 36601

Name: Word Processor Program
System: PET
Memory: Not Specified.
Language: Not Specified.
Hardware: RS-232 printer addressed via a CmC printer adapter.
Description: This program permits composing and printing letters, flyers, advertisements, manuscripts, articles, etc., using the Commodore PET and an RS-232 printer. Script directives include line length, left margin, centering, and skip. Edit commands allow the user to insert lines, delete lines, move lines, change strings, save onto cassette, load from cassette, move up, move down, print and type.
Copies: Not Specified.
Price: $29.50
Ordering Info: None.
Author(s): Not Specified.
Available from:
 Connecticut microComputer
 150 Pocono Road
 Brookfield, CT 06804

Name: ZIP TAPE
System: KIM-1, may be easily modified for any other 6502 system with programmable timer I/O
Memory: 3/4 page each for read and write progs.
Hardware: Simple single IC audio to logic level converter and output buffer/attenuator on 2" sq. board. Directional control, 4 connections to computer.
Description: A fast audio cassette data recording and recovery system. Programmable to 4800 baud. Loads 8K in less than 15 seconds. Follows KIM-1 protocol of open ended record length with start address, end address, and record ID specified at usual KIM locations. Load by ID, ignore ID, and relocate modes. Data recorded in binary form with 2 byte checksum error detection. Easily relocated, can either stand alone or be used as subroutines. Requires programmable timer I/O.
Copies: About 12, just introduced.
Price: $22.50 +1.00 ship & hand. $3.00 extra for KIM cassette.
Includes: Assembled and tested interface, commented listings, suggested changes to run on TIM and other systems. Cassette has software recorded at HYPERTAPE and standard KIM speeds plus 8K test recording using ZIP TAPE.
Ordering Info: With or Without tape.
Author: Lewis Edwards, Jr.
Available from:
 Lewis Edwards
 1451 Hamilton Avenue
 Trenton, NJ 08629

Name: FOCAL* (*DEC Trademark)
System: Apple II
Memory: Not Specified.
Language: Assembler
Hardware: Apple II
Description: This is an extended version of the high-level language called FOCAL. FOCAL was created for the DEC PDP-8. It is similar to BASIC. FCL65E, as this version is called, is now available for the Apple II.
Copies: Not Specified.
Price: Apple II format cassette - $25.00
 Mini-Manual - $6.00
 FCL65E User's Manual - $12.00
 Complete Source Listing - $35.00
Ordering Info: Specify parts desired.
Author(s): Not Specified.
Available from:
 The 6502 Program Exchange
 2920 Moana
 Reno, NV 89509

Name: WARLORDS
System: Apple II (PET version under devel.)
Memory: Not Specified
Language: Not Specified
Hardware: Apple II
Description: It is the Dark Ages, in the kingdom of Nerd, and all is chaos. King Melvin has died without an heir and a dire power struggle is taking place to see who will emerge as the new King. You and the other players are the WARLORDS, and you will have to decide what combination of military might and skillful diplomacy will lead you to victory.
Copies: Not Specified
Price: $12.00
Ordering Info: Specify Apple II Version
Author: Not Specified
Available from:
 Dealers who carry software from
 Speakeasy Software LTD.

Names: E/65 and A/65
System: Any 6502 based system
Memory: Not Specified
Language: Assembly
Hardware: Terminal. Cassette optional.
Description: E/65 is primarily designed to edit assembler source code. Line oriented commands specify input/out or text and find specific lines to be edited. String oriented commands allow the user to search for and optionally change a text string. Also character oriented commands and loading and dumping to bulk device. A/65 is a full two-pass assembler which conforms to MOS Technology syntax. A full range of run-time options are provided to control listing formats, printing of generated code for ASCII strings and generation of object code.
Copies: Not Specified
Price: $100 each
Includes: Object form on paper tape or KIM type cassette. Listings of source code are available for $25.00 each. Full documentation on the installation and use of each package is provided.
Author: Not Specified
Available from:
COMPAS - Computer Applications Corporation
P.O. Box 687
Ames, IA 50010

Name: Read/Write PET Memory
System: PET
Memory: 8K RAM
Language: BASIC
Hardware: Standard PET
Description: Permits user to key into memory hex codes by typing hex starting address and then typing the hex digits in sequence desired. Display memory as both hex codes and assembly language mnemonics (translates relative address into actual hex address). Stores memory on tape and loads memory from tape into any desired memory location. Executes machine-language programs.
Copies: Just released - 32 sold first day.
Price: $7.95 - postpaid
Includes: Cassette tape; complete instructions (including use of ROM subroutines to input and output memory from keyboard and to screen).
Ordering Info: From author
Author:
Don Ketchum
313 Van Ness Avenue
Upland, CA 91786

(Dealer Inquities Invited)

---

The MICRO Software Catalog is a continuing feature of MICRO. If you have any 6502 based software for sale (or exchange or free), please send a complete description which includes ALL of the information listed.

The MICRO Staff will not write up entries for the MICRO Software Catalog from other materials that you may provide. First, we do not have the time to do this. Second, since we are not as familiar with your software as you are, we can not hope to provide as meaningful a write-up as you can. Cover all pertinent information, but keep the write-up to a reasonable length. MICRO reserves the right to reject or edit any material submitted for this column.

Name of program:
6502 systems:
Memory locations required:
Language (BASIC, Assembler,...):
Hardware required:
Description of program:
Number of copies sold to date:
Price:
What is included in package (cassette, listings, paper tape, ...):
Ordering information:
Author(s):
Company Name and Address:

Send to:
MICRO, P.O. Box 3, S. Chelmsford, MA 01824

## THE MICRO HARDWARE CATALOG

### A Call for Information

Starting with the next issue of MICRO, we plan to run a Hardware Catalog similar to the current Software Catalog. Information for this catalog will come from suppliers of the hardware: the manufacturer, distributor or dealer. This will NOT be a "Product Review" nor will inclusion of information indicate endorsement of the product by MICRO. We will not knowingly include products which do not meet the following guidelines:

1. The product must be directly related to 6502 interests. For example, a general purpose coding form would not qualify.

2. The product must be currently available:
   A. Some units must have already been delivered.
   B. Delivery on new orders should be no more than stock to four weeks.

3. The price must be included, along with any other pertinent information about discounts, shipping charges, etc.

Suggestions for Hardware Catalog information:

1. Cover all of the important features of your product, but be concise. MICRO reserves the right to edit submissions which are too long.

2. A "picture is worth a thousand words" and doesn't cost you a thing. Since it is a lot more work to include pictures in the catalog, we are not sure that we will be able to use them, but if it is possible, we will.

3. Submit separate products as separate items for the catalog. First, we will not print conglomerate listings. Second, you get multiple exposure with separate listings.

4. Don't waste your time or ours submitting material which does not directly relate to the 6502 family.

MICRO reserves the right to reject any item submitted for inclusion in this catalog.

# A DEBUGGING AID FOR THE KIM-1

Albert Gaspar
305 Wall Street
Hebron, CT 06248

DEBUG is a program designed to assist the user in debugging and manipulating programs. It resides in memory locations 1780 - 17E6 and provides a means for inserting breakpoints in a user program, moving blocks of bytes throughout memory, filling memory with repetitious data, and calculating branch values. It uses selected KIM monitor subroutines.

## Operating Modes

DEBUG has three operating modes:

1. Keyboard Mode: DEBUG remains in a wait loop anticipating keyboard entry which will be recognized as either data or command characters. This mode is initiated either by using the KIM monitor to start at location 178E, or by the execution of a previously inserted breakpoint in a user program.

2. Execute Mode: DEBUG executes logic to service a user command. This mode is completed in microseconds and will not be noticeable by the user.

3. Non-Control Mode: DEBUG relinquishes control when the user keys in "RS", or "ST" during Keyboard Mode, or uses the CONTINUE Command.

To start, the user must first load "B5" into 17FE and "17" into 17FF using the KIM. Then the user begins DEBUG by starting at location 178E. This puts DEBUG into Keyboard Mode. The user then keys in combinations of the 16 data characters available on the keyboard. Input data is displayed in a manner similar to that of the KIM - from right to left - except that only the left-most five display positions are utilized (exceptions are noted below).

The user must continue to key in characters until he is satisified that the required data is input. Then one of the several Command code characters available (B, C, D, E, or F) is keyed in. At this point, or at any time previous to this, if the input is not correct and the user wishes to change the display, he merely continues to enter data until the display string is correct. When the display concatenation is satisfactory (either 2 or 4 data characters and 1 Command character) he keys in "AD". Now DEBUG will go into Execute Mode (without echoing the entry of "AD") and immediately examines the last previous character input. If this character is not a legitimate Command character (B, C, D, E, or F), DEBUG becomes confused and will transfer to unpredictable memory locations. Thus the user is held wholly responsible for the validity of his input. He should always check that either his keyed-in data is correct before hitting "AD", or that his Command was indeed executed. Note: if a key other than "AD", the 16 data characters, "RS", or "ST" is depressed, its high order 4 bits are stripped and the remaining low order 4 bits are displayed and evaluated as whatever the combination happens to represent.

Assuming that the character input immediately prior to "AD" is a legitimate Command character, DEBUG - still in Execute Mode - will process the data which was input prior to the Command code (either 2 or 4 characters). Note that the Command values (B, C, D, E, of F) if found in the data field are processed as standard hex values.

**BREAK** This command allows the user to insert a breakpoint anywhere desired in his program. When this point is subsequently reached during execution of his program, control will be passed to Keyboard Mode of DEBUG and further execution of the user program will effectively be temporarily discontinued. Also at this time the user area will be restored to the original configuration existing at the time of the breakpoint insertion.

Input Sequence:

| Press Keys | See on Display |
|---|---|
| 4 Data Characters B "AD" | 4 char B1 |

The 4 Data Characters define the Breakpoint location desired. The BREAK Command saves the user byte at the Breakpoint and deposits a BRK instruction in place of it. Thus, that user area should not be altered by the user while DEBUG is in Non-Control Mode and a Breakpoint is eminent, or the Breakpoint return will not work. More than one Breakpoint can be eminent at one time; however since DEBUG will store only one byte at a time, multiple simultaneous Breakpoints should be applied only at user locations containing the same instruction. This way it is immaterial which BRK triggers a return to DEBUG - the user area will be properly replaced.

This Command includes 1 of 2 instances where the sixth display position is used. If the sixth position contains a 1, the Command has been correctly processed. If the position contains any other value, it indicates that depression of the "AD" key has caused multiple bounces and the byte stored by DEBUG within itself is now "00" - not the original user byte. Thus DEBUG will still function correctly but will not correctly restore the user position when a Breakpoint return is initiated. The user must restore the location manually (using KIM) after the return has been performed - otherwise "00" will be left in the location.

**CONTINUE** This Command causes DEBUG to pass execution to a user specified location. It is similar to the passing of control through KIM and either method may be used to execute user code.

Input Sequence:

| Press Keys | See on Display |
|---|---|
| 4 Data Characters C "AD" | 4 char C0 |

The 4 Data Characters define the address to which control is to be passed. The above display is only momentary since control is immediately passed to a user area (Non-Control Mode) The purpose of the Continue Command will usually be to execute to a previously inserted Breakpoint. When this occurs, as previously stated, control returns to Keyboard Mode, of DEBUG. At this point, the leftmost 4 display digits will contain the address at which the Breakpoint was located. See Overall Notes #1 for a continuation warning.

**MOVE** This Command will move a block of up to 256 bytes to another memory area. It is non-destructive (unless, of course, a shift is performed).

Input Sequence:

| Press Keys | | | See on Display | |
|---|---|---|---|---|
| 4 Data Characters | F | "AD" | 4 char | F0 |
| | (F for From) | | | |
| 4 Data Characters | D | "AD" | 4 char | D0 |
| | (D for Destination) | | | |
| 2 Data Characters | E | "AD" | XX 2 char | E0 |
| | (E for Execute) | | | |

The 4 Data Characters above represent the locations one less than the locations, respectively, from which and to which the data is to moved. The 2 Data Characters above represent the hex value of the number of bytes to be moved. If the user desires to move 256 (dec.) bytes, he must input "00" in the "E" Command. "F" and "D" execution may be input in either order - "F" then "D" or "D" then "F".

MOVE will correctly move blocks of bytes from one area of memory to another. However it will correctly shift bytes only in an upward direction. Attempting downward shifts will result in the repeating of as many of the last bytes in the original block as there is a difference in the block positions. For example - shifting a block of say (n) bytes starting at 0200 to a new area starting at 0202 will correctly shift the (n) bytes upward 2 locations. Attempting to shift a block of (n) bytes starting in 0202 to a new area starting in 0200 will result in the last 2 bytes of the original block to be repeated downward from their original locations continuing to 0200. This may not be completely undesireable since - 1) normally the user will be interested in expanding an area, not in compressing it (for example, to add instructions); and, 2) this serves as a useful tool to provide filler bytes in memory when desired.

BRANCH This Command assists in calculating Branch values.

Input Sequence:

1. Enter the necessary 12 bytes of Branch Overlay, either through KIM or by tape overlay. (These will, of course, have to be restored to the original configuration when through with BRANCH).

1. Put DEBUG into Keyboard Mode.

| Press Keys | | | See on Display |
|---|---|---|---|
| 2 char/2 Char. | E | "AD" | 2 char/2 char/D-VALUE |

The first 2 characters are the 2 least significant values of the Branch Address. The next 2 characters are the 2 least significant values of the Branch to Address. The "E" stands for Evaluate. The correct Displacement VALUE will appear in the 5th and 6th display positions. The displacement is calculated assuming that the two addresses are in the same page. For page overlap, entry will have to be done twice. We believe that different users will have different preferential methods for doing this, so our own method, which is somewhat involved, is not described. If both entries are on the same page but are separated by a distance greater than the standard branch range, the value calculated will be incorrect. It is the user's responsibility to check for out-of-range values.

Overall Notes

1. When a Breakpoint has been executed, DEBUG does not store and then restore accumulator, register, and status values. Thus, the user must take care in continuing from a Breakpoint if any of these parameters have a subsequent bearing in further user program execution. (Though this and other omissions are glaring defects, no apology is made - there was just insufficient memory available for inclusion of any refinements.)

2. When returning from a "BRK" instruction, DEBUG pulls the status register information from the stack and ignores it. If this DEBUG version is used in conjunction with an interrupt system, locations 17FE - 17FF must contain the address of the user interrupt handler. The beginning of the handler must be similar to that shown on page 144 of the KIM Programming Manual. The logic listed in example 9.7 must be utilized as shown. "BNE BRKP" will point to the DEBUG location defined below. If the user handler determines that the interrupt was caused by "BRK", then the handler must jump to location 17B5. DEBUG will then obtain the "BRK" address and perform subsequent logic to return the user byte to its original configuration and continue on into Keyboard Mode.

3. This version of DEBUG uses page zero locations 0000, 0001, 0002, 0003, and 0004, but only as scratch areas during Keyboard and Execute Modes. The user can use these areas as temporary scratch areas when DEBUG is not being executed.

4. Due to limited instruction space, DEBUG is particularily susceptible to key bounce. The user should remain watchful of such occurrences, especially during BREAK execution as previously described.

5. My goal here was to fit as much DEBUG power into locations 1780 - 17E6 as possible - not to write a great breakpoint/move/branch calculate routine. (That has already been done by others) Thus DEBUG had to be written in relatively concise and tight code, using data as instructions, instructions as data, overlapping instructions, using the same code to do different things, instruction modification, position instructions in prescribed relative locations, use of "write-only-memory", etc. I do not approve of this type of programming - in fact I strongly recommend against it. However, in this case I hope the goal I had justifies the mess that DEBUG has turned out to be. In any event I would like to point out that as tight as the code is, it is still possible to add other functions here and there. For example the version I usually use displays the value of the accumulator in display locations 5 and 6 when returning back from a Breakpoint. At times I also use another version which doesn't require the "BRK" instruction at all. This is convenient when debugging interrupt programs since no additional interrupt is needed for DEBUG. However, both versions penalize me in other areas, which makes it all a trade-off decision.

[Editor's Note: Gaspar seems to be suggesting a collection of specialized DEBUG programs, each customized to provide a particular set of capabilities while residing in minimal memory. Using his code as a starting point, a "programwise" reader should be able to construct his own set of DEBUG aids.]

```
         ZERO   *      $0000  LOCATION 0000
         ONE    *      $0001
         TWO    *      $0002
         THREE  *      $0003
         FOUR   *      $0004

         INH    *      $00F9  KIM DISPLAY POINTERS
         POINTL *      $00FA
         POINTH *      $00FB

         RETURN *      $17B5  INTERNAL ADDRESS
         TBLOFF *      $17D4  TABLE OFFSET
         JUMPER *      $17DD  INTERNAL ADDRESS

         INITI  *      $1E8C  KIM INITIALIZE ROUTINE
         SCANDS *      $1F1F  KIM SCAN DISPLAY ROUTINE
         GETKEY *      $1F6A  KIM GET KEYBOARD CHARACTER
1780 B1 02      EXEC   LDAIY TWO    GET CHAR TO BE MOVED
1782 91 00             STAIY ZERO   MOVE IT
1784 88                DEY
1785 D0 F9             BNE   EXEC   CONTINUE UNTIL DONE
1787 98         DANDF  TYA          GET TO OR FROM ADDRESS
1788 95 F3             STAZX $00F3  STORE IT IS SCRATCH
178A A5 FB             LDAZ  POINTH
178C 95 F4             STAZX $00F4
178E 20 8C 1E   START  JSR   INITI  SET FLAGS AND INIT.
1791 20 1F 1F          JSR   SCANDS DISPLAY BUFFER
1794 D0 F8             BNE   START
1796 20 1F 1F   KEY    JSR   SCANDS NEW CHARACTER INPUT?
1799 F0 FB             BEQ   KEY    NO, CONTINUE TO DISPLAY
179B 20 6A 1F          JSR   GETKEY YES, GET THE CHARACTER
179E A6 04             LDXZ  FOUR   PICK UP LAST CHAR. INPUT
17A0 C9 10             CMPIM $10    IS THE NEW CHAR. "AD"?
17A2 F0 30             BEQ   PROCES YES.  PROCESS CURRENT COMMAND
17A4 85 04             STAZ  FOUR   NO. STORE IT
17A6 A2 04             LDXIM $04    AND SHIFT IT INTO THE DISPLAY
17A8 0A         SHIFT  ASLA
17A9 26 F9             ROL   INH    SHIFT THE DISPLAY LEFT
17AB 26 FA             ROL   POINTL
17AD 26 FB             ROL   POINTH
17AF CA                DEX
17B0 D0 F6             BNE   SHIFT  DONE SHIFTING
17B2 85 F9             STA   INH    YES.  ADD NEW CHAR TO DISPLAY
17B4 F0 D8             BEQ   START  UNCONDITION RETURN
17B6 38                SEC
17B7 68                PLA          IGNORE STATUS
17B8 68                PLA          GET "FROM" ADDRESS
17B9 E9 02             SBCIM $02    SUBTRACT 2
17BB 85 FA             STAZ  POINTL DISPLAY LOW ORDER
17BD 68                PLA
17BE E9 00             SBCIM $00    SUBTRACT CARRY, IF ANY
17C0 85 FB             STAZ  POINTH DISPLAY HI ORDER
17C2 A2 0C             LDXIM $0C    CHEAT ON RX
17C4 E6 F9      B      INC   INH    COUNT KEY BOUNCES
17C6 A0 00             LDYIM $00
17C8 B1 FA             LDAIY POINTL GET USER BYTE
17CA 9D DC 17          STAX  $17DC  STORE IT
17CD BD DB 17          LDAX  $17DB  GET "BRK"
17D0 91 FA             STAIY POINTL STORE IN USER AREA
17D2 A2 0D             LDXIM $0D    CHEAT ON RX
17D4 A4 FA      PROCES LDYZ  POINTL
17D6 BD D4 17          LDAX  TBLOFF PREPARE TO GO TO COMMAND LOGIC
17D9 8D DD 17          STA   $17DD  ALTER INSTRUCTION
17DC D0 FF             BNE   JUMPER JMP TO COMMAND LOGIC
17DE EA                NOP          FUTURE EXPANSION
17DF E6         TABLE  =     $E6    BRANCH TO "B"
17E0 06                =     $06    BRANCH TO "C"
17E1 A9                =     $A9    BRANCH TO "D"
17E2 A2                =     $A2    BRANCH TO "E"
17E3 A9                =     $A9    BRANCH TO "F"
17E4 6C FA 00   C      JMI   POINTL 00 OR ADDRESS USED AS "BRK"
```

## BRANCH CALCULATION OVERLAY

```
                              ORG    $1780

                       INH    *      $00F9
                       POINTL *      $00FA
                       POINTH *      $00FB

1780 38                EXEC   SEC           INITIALIZE SUBTRACT
1781 A5 FA                    LDAZ   POINTL
1783 69 FD                    ADCIM  $FD    CORRECTION CONSTANT
1785 E5 FB                    SBCZ   POINTH
1787 85 F9                    STAZ   INH    STORE RESULT IN DISPLAY
1789 4C 8E 17                 JMP    $178E  JUMP TO START
```

### Examples

1. Load DEBUG. Load "B5" into 17FE and "17" into 17FF.

2. Start execution at location 178E.

3. Depressing any of the 16 keyboard characters will cause the 5 leftmost display digits to shift left and the new character to be inserted into the fifth position.

4. Assume that there is a program in 0200-0250. Now, to execute from 0200-0240:

```
0 2 4 0 B AD        Display is  0240 B1
0 2 0 0 C AD                    0200 C0
                                0240 XX
```

When the user program executes to location 0240, it will return to DEBUG which then will replace the original byte at 0240 and will return to Keyboard Mode.

5. User wishes to add a 3 byte instruction in 0241-0243. Thus he must shift his program from 0241-0250 to 0244-0253.

```
0 2 4 0 B AD        Display is  0240 B1
0 2 4 0 F AD                    0240 F0
```

(Remember that MOVE requires addresses 1 less than the actual values.)

```
X X 1 0 E AD        Display is  XX10 E0
```

(10 = 0250 - 0241 + 1)

This shifts bytes in 0241-0250 to 0244-0253. User can now insert his 3 new instructions into locations 0241, 0242, and 0243.

6. User wishes to load NOP into locations 0300-03FF. Load "EA" into 03FF using KIM. Return to DEBUG.

```
0 3 0 0 F AD        Display is  0300 F0
0 2 F F D AD                    02FF D0
    0 0 E AD                    XX00 E0
```

(Move 256 decimal bytes.)

7. User wishes to calculate the value required for a HERE BCC START where HERE = 0204 and START = 0250.

First, load overlay (12 bytes) and return to DEBUG.

```
0 4 5 0 E AD        Display is  0450 4A
```

Thus the branch value is 4A and the branch instruction will be BCC 4A.

Remember that if further DEBUG usage is planned, the original 12 bytes starting at 1780 have to be replaced.

### Program Notes

1. The instruction listings at 17B4 and 17E4 are NOT errors and must be placed in memory exactly as shown.

2. Locations 17E7 and 17E8 are used by the KIM monitor for tape checksum. However, their usage in DEBUG will not interfere with KIM since the two programs do not, of course, use them at the same time.

# 6502 INTERFACING FOR BEGINNERS: ADDRESS DECODING II

Marvin L. De Jong
Dept. of Math-Physics
The School of the Ozarks
Point Lookout, MO 65726

I hope you did not turn any expensive integrated circuits into cinders with last month's experiments. We will begin this month by considering the questions raised in the last column. You will need to refer to the circuits, tables, and the program described there. The following table describes the activity which takes place on the address bus and the data bus while the program is running. It is organized by clock cycles, each one microsecond long, starting with the op code fetch of the CLC instruction.

| CYCLE | ADDRESS BUS | A15 | A14 | A13 | DATA BUS | COMMENTS |
|---|---|---|---|---|---|---|
| 0 | 0200 | 0 | 0 | 0 | CLC op code | Pin 1 of LS145 is low because address lines A13-15 are low. |
| 1 | 0201 | 0 | 0 | 0 | STA op code | LED will glow when connected to pin 1, but not to other pins. |
| 2 | 0201 | 0 | 0 | 0 | STA op code | All other pins on LS145 are high. |
| 3 | 0202 | 0 | 0 | 0 | XX | Low order address of storage location on data lines. |
| 4 | 0203 | 0 | 0 | 0 | 60 | High order address of storage location on data lines. |
| 5 | 60XX | 0 | 1 | 1 | accumulator contents | LED will light for 1 microsecond if connected to pin 4 on LS145. |
| 6 | 0204 | 0 | 0 | 0 | BCC op code | Pin 4 high, pin 1 low. LED will glow on pin 1 only. |
| 7 | 0205 | 0 | 0 | 0 | FB offset | 6502 is now determing if and where to branch. Branch is to 0201 because |
| 8 | 0206 | 0 | 0 | 0 | garbage | carry was clear. |

In the program loop address lines A14 and A13 go high only during cycle 5. Thus, for six cycles output 0 (pin 1) of the LS145 is low. The LS145 is an open collector device and acts like a switch to ground when the pin is in the L state, allowing current to flow through the LED. During cycle 5, when the address of the storage location is on the address bus, pin 4 is in the low state and will cause the LED to glow. Earth people do not perceive one microsecond flashes spaced six microseconds apart, so the LED appears to glow rather than flash. Since the majority of the loop time is spent with pin 1 at logic 0, a bright glow is observed on this pin. Changing the instruction from STA to LDA has no effect since the address bus goes through the same sequence for a LDA as it does for a STA. Changing the storage location from 60XX to something else will cause another pin of the LS145 to glow. The results of the LED test should agree with the truth table given for the LS145.

The pulse from the decoder which occurs when it responds to a particular address at its input pins is called a device select pulse or an address select pulse. The LS145 produces a logic 0 or active-low device <u>se</u>lect pulse, sometimes symbolized by ⌐∟⌐ or $\overline{DS}$. This pulse is used to select or activate or enable another device in the computer system such as a memory chip, an I/O port, a PIA chip, or another decoder. As mentioned in the last column, the device select pulse from the LS145 could be used to enable a 74LS138 which would then decode address lines A10-12, dividing an 8K block into 1K blocks. Such a scheme is very similar to the expansion circuit suggested in the **KIM-1 USER MANUAL**, page 74. Similar circuits are also used on memory expansion boards. In the present circumstance I have decided to make a trade-off between wasting address space and minimizing the number of chips on the breadboard. Our purpose here is to configure some I/O ports as simply as possible.

The decoding circuit is shown in Figure 1. A total of eight device select pulses are available for eight I/O ports. Note that one of the 8K selects (8K4) from the LS145 enables the LS138 which decodes the three low-order address lines. All of the 8K4 space is used to get eight I/O ports. Using a 74LS154 instead of the LS138 and decoding on more address line would give 16 I/O ports in the event we need more. Or we could take another 8K select to enable another LS138 or LS145, giving us 8 or 32 ports, respectively. There is no doubt that address space is being wasted, but few users use all 64K, or even 32K, so the waste may be justified. In Figure 1, address lines A0-2 are extended downward to indicate that they could be decoded by other devices such as an LS138 or LS154.

The addresses which enable the device select pulses $\overline{DS0-7}$ are given in Figure 1. Note that since not all sixteen lines have been decoded to produce the pulses, the addresses shown are not the only ones which will work. For example, device select pulse 0 will be produced whenever the computer reads or writes to 8XX0 or 9XX0 (XX means any hex numbers). This should cause no difficulty unless we try to put other devices into the 8K4 block, in which case we could simply decode some other lines. If your system does not buffer the address lines, you should buffer them with the circuit shown in Figure 2.

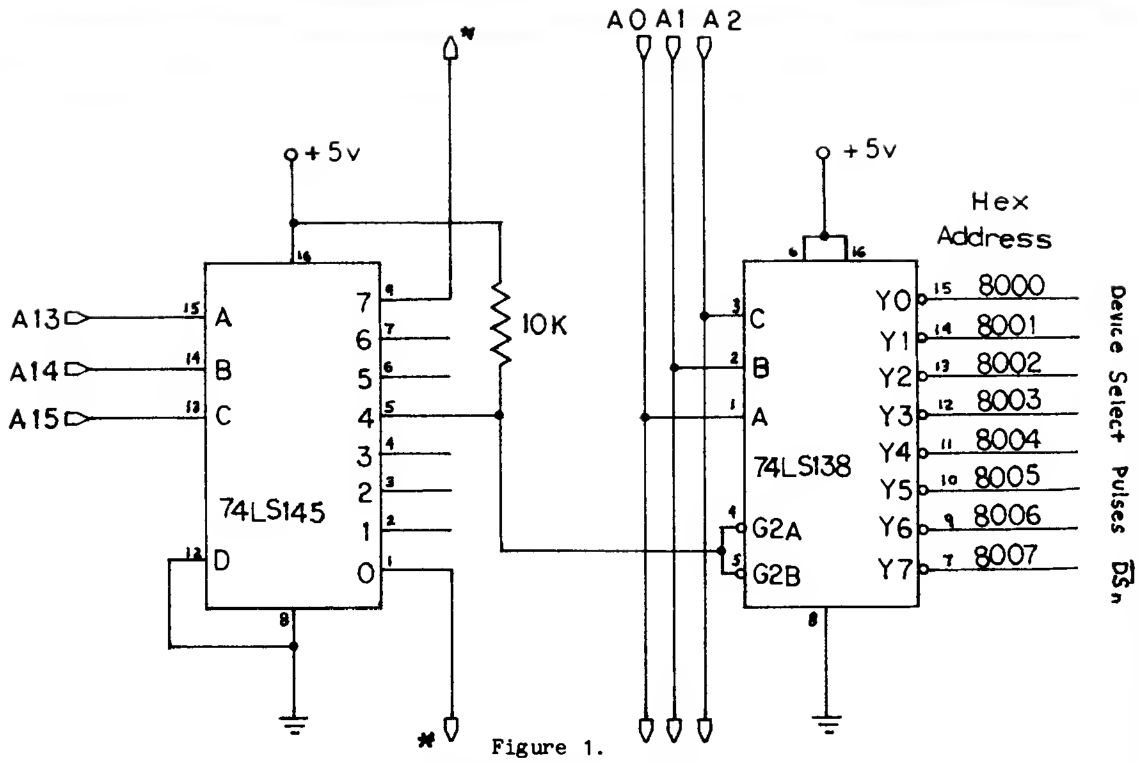



Figure 1.

Decoding Circuit to Select I/O Ports.
* See text for details.

Construct the circuits of Figures 1, 2, and 3. I managed to get them on one A P circuit board with no difficulty, with room for several more chips. I also found that the A P breadboard jumper wire kit is very handy for making neat layouts. Connect one of the device select lines from the LS138 to the flip-flop preset input (Test Circuit, Figure 3) and another device select line to the clear input. A pulse to the preset input will cause the Q output to go high, lighting the Q LED, whereas a pulse to the clear input will cause the $\overline{Q}$ output to go high, lighting the $\overline{Q}$ LED.

To test your decoding circuit write a one statement program, for example:

```
0200   AD 00 80        LDA DS0
```

If the line labeled 8000 is connected to the preset of the test circuit, the Q output will go high, lighting the LED, when the program is run. Running the program:

```
0200   AD 04 80        LDA DS4
```

will cause a switch of the flip-flop if the line 8004 is connected to the clear input. You should test all 8 device select lines from the LS138 with these programs by changing the connections and the addresses. Note that no data is being transferred since we have made no connections to the data bus. It should also be apparent that this scheme could be used to switch a motor, light, cassette recorder or other device off and on in a computer program. Eureka! We have made a simple I/O circuit.

To continue a little further, repeat the above experiments with a STA instruction replacing the LDA instruction. The results should be identical because in both cases it is the address of the device select on the address bus which produces the pulse which flips the flop. One more experiment: connect the R/W line from the 6502 to the G1 input on the LS138 after removing the connection from G1 (pin 6) to pin 16. Now try the programs above, using first a LDA instruction, then a STA instruction. You should find that the program with the LDA instruction

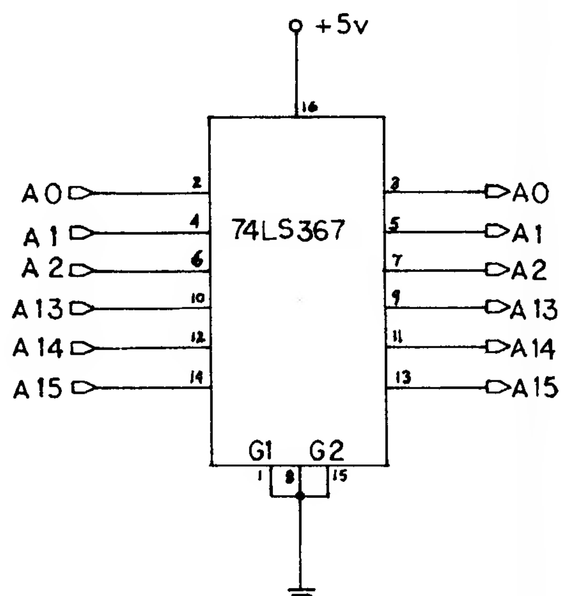



Figure 2.

Buffering the Address Lines.
The arrows pointing into the chip are the lines from the 6502, while those pointing away go to the circuit in Figure 1.

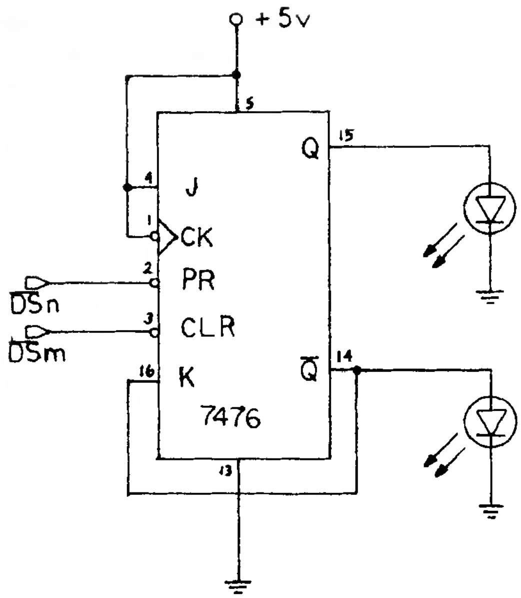



Figure 3. Test Circuit.

works, that is, the lights can be switched from off to on and vice versa, but the STA instruction does not work. Why?

Keep your circuit, as the material in the next column will refer to and make use of the circuit you have just completed.

A Note About Figure 1: The * lines in Figure 1 suggest that something should be done with them. For the experiments described above, nothing need be connected to these lines, however when we try to put data on the data bus these lines will become important. What you do depends on the system you are using. Since the KIM-1 is probably the most popular system among the readers, and since my own system is a KIM (expanded with a Riverside KEM and MVM-1024) the following details will be of most interest to KIM owners. Owners of other systems will have to dig into their manuals to make sure they are not de-selecting their on-board devices, or much worse, selecting two devices to put information on the data bus simultaneously. The KIM-1 has a 74145 decoder on-board which decodes lines A10-12; lines A13-15 are not decoded. Consequently, the lowest 8K0 block is already decoded, and the device select pulse from the LS145 in Figure 1 should enable the decoder on the KIM for all addresses in the 8K0 block. To do this simply connect the device select pulse from pin 1 on the 74LS145 in Figure 1 to pin K on the application connector on the KIM, making sure that the ground connection is first removed. A 10K pull-up resistor between pin 1 and +5V will also be necessary. The device select pulse from 8K7 should enable the device containing the restart and interrupt vectors. In the case of the KIM, pin 9 of the LS145 in Figure should enable the 6530-002 ROM by connecting it to pin J of the application connector. No pull-up is necessary.

Next issue we will examine the other pins on the 6502 which will be useful in configuring I/O ports, namely the bi-directional data bus, and the control signals. Hopefully we shall finish the circuitry needed to make an output port (8 bits), connect some LEDs to it, see if it works or smokes, and maybe think of a use for it.

A couple of parting shots: First, there is a very good educational series of articles in KILOBAUD magazine called KILOBAUD KLASSROOM. It assumes less experience than I have assumed so far. Second, I hope you have obtained a "TTL Databook" from either Texas Instruments or National so that you can study the truth tables and other specifications of the chips we are using.

## An Additional Experiment

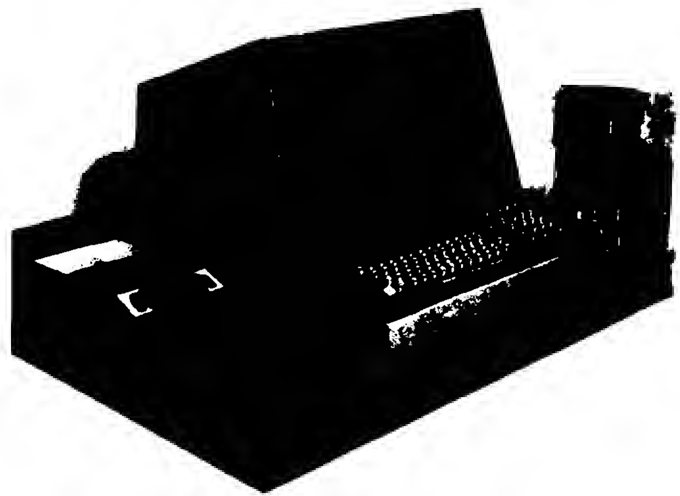
The address decoding circuit of Figure 1 produces a one microsecond negative going one-shot pulse when a LDA instruction addresses one of the locations shown in Figure 1. This one-shot can be used for a variety of purposes, one of which is triggering the flip-flop shown in Figure 3. The program listed below makes use of an interval timer (KIM-1 system addresses) to produce a square wave. By varying the time loaded into the timer, the frequency can be changed, and the duty cycle can be changed. Thus, we have produced a simple function generator with programmable period and duty cycle. The LEDs will show the results at low frequencies. Try this program and watch the LEDs. Amplify the Q output and connect it to a speaker; notice the effect of changing the time, the duty cycle, the wave shape (by filtering) or whatever else you can think of. Notice that I used device selects 8007 and 8001.

```
                DSEVEN *     $8007  DEVICE SELECT 7
                DSONE  *     $8001  DEVICE SELECT 1
                TIMER  *     $1707  KIM TIMER
                CLKRDI *     $1707  KIM CLOCK DONE TEST

0200 AD 07 80   START  LDA   DSEVEN INIT DS7 DEVICE SELECT PULSE
0203 A9 FF             LDAIM $FF    INIT TIMER
0205 8D 07 17          STA   TIMER  START DIVIDE-BY-1024 TIMER FOR 256
0208 AD 07 17   BACK   LDA   CLKRDI CYCLES, NOW CHECK TO SEE IF IT
020B 10 FB             BPL   BACK   IS FINISHED.  IF NOT, CHECK AGAIN,
020D AD 01 80          LDA   DSONE  OTHERWISE TRIGGER DS1.
0210 A9 FF             LDAIM $FF
0212 8D 07 17          STA   TIMER  START TIMER FOR SECOND HALF OF
0215 AD 07 17   AGN    LDA   CLKRDI CYCLE.  IS TIMER READY?
0218 10 FB             BPL   AGN    NO, CHECK AGAIN, OTHERWISE JUMP
021A 4C 00 02          JMP   START  TO START OVER.
```

# Introducing the KIM BETA . . .

## a system expansion of the most popular single board computer to its full potential!



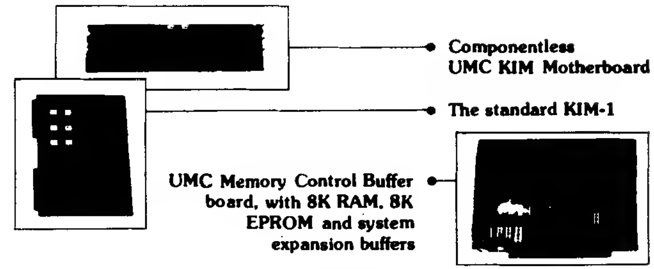
The KIM BETA is the HEART of a Full-Scale Computer System, including disk, printer, modems.

Suitable for. . . education. . . research. . . industrial control. . . and many other applications.

Available from: United Microsystems Corporation
"The Professional Computer Store"™
2601 South State Street
Ann Arbor, Michigan 48104
(313) 668-6806

## The KIM BETA offers unique expansion capabilities . . .

***A unique systems approach which allows immediate expansion to 64K, beginning with 8K RAM and 8K EPROM with your choice of BASIC or FOCAL in EPROM or cassette tape, via the KIM-1 on-board cassette interface, and you use economical S-100 static memories for immediate expansion.***




## KIM BETA Systems offer multiple expansion slots for:

- Economical S-100 memories
- DMA controller for MINI-FLOPPY DISKS
- EPROM programmer with personality modules
- Analog to Digital, Digital to Analog, or Modems boards
- Video-graphic display controller
- IEEE 488 BUS ADAPTOR
- Adding your own interface or WHATEVER
- A memory bank switcher
- Full keyboard including numeric keypad and function keys so you can have full control of your KIM-1 without modifying it
- And your KIM is COMPLETELY BUFFERED FROM ADDRESS AND DATA BUSSES TO AVOID OVERLOADING

The basic system price of the KIM BETA including Memory Control Buffer board, 8K RAM, and Componentless Motherboard with 12" CRT, Full Keyboard with numeric pad and power supply.

The KIM BETA will arrive all enclosed in its own attractive cabinet, completely assembled and tested.

only **$995.00**

*(all software optional and available)*

United Microsystems Corporation
"The Professional Computer Store"™
2601 South State Street
Ann Arbor, Michigan 48104
(313) 668-6806

# BROWN AND WHITE AND COLORED ALL OVER

Richard F. Suitor
166 Tremont Street
Newton, MA 02158

This article consists of two parts. The first is a brief discussion of the colors of the Apple and their relationships to each other and to the color numbers. Some of that information is used in the second part to generate a random color display according to certain principles suggested by Martin Gardner in his mathematical games column in Scientific American.

## The Color of Your Apple

The color of your Apple comes from your color TV. The video signal has many components. Most of the signal carries the brightness information of the picture - a black and white set uses this part of the signal to generate its picture. Superimposed on this signal is the "color carrier:, a 3.58 MHz signal that carries the color information. The larger this signal, the more colorful that region of the picture. The hue (blue, green, orange, etc.) is determined by the phase of the color signal. Reference timing signals at the beginning of each scan line synchronize a "standard" color signal. The time during a 3.58 MHz period that the picture color signal goes high compared to when the standard goes high determines the hue. A color signal that goes high when the standard does gives orange. One that goes low at that time gives blue. Signals that are high while the standard goes from high to low or from low to high give violet and green. (This, at least, was the intention. Studio difficulties, transmission paths and the viewers antenna and set affect these relations, so the viewer is usually given final say with a hue or tint control.)

The time relation of the color signal to the standard signal is expressed as a "phase angle", is measured in angular measures such as degrees or radians and can run from 0 to 360 degrees. This phase angle corresponds to position on a color circle, with orange at the top and blue at the bottom, as shown in Figure 1.

The perimeter of the circle represents different colors or hues. The radial distance from the center represents amount of color, or saturation. The former is usually adjusted by the tint control, the latter by the color control. A color that can be reproduced by a color TV can be related to a point in this circle. The angular position is coded in the phase of the 3.58 MHz color carrier signal; the radial distance from the center is given by the amplitude of the color carrier.

The numerical coding of the Apple colors can be appreciated using this circle and binary representation of the color numbers. The low order bit corresponds to red (#1). The second bit corresponds to dark blue (#2), the third to dark green (#4) and the high order bit to brown (dark yellow, #8). To find the color for any color number, represent each 1 bit as a quarter-pie piece centered over its respective color, as indicated in Figure 1. The brightness or lightness of the color corresponds to the number of pie pieces and the color corresponds to the point where the whole collection balances. Black, #0, has no bits set, no pie and no brightness. White, #15, has four bits set, the whole pie, is of maximum brightness and balances in the center of the circle at neutral. Orange, #9 or 1001 in binary, has pie over the top hemisphere and balances on a point between neutral and orange. The #5, binary 0101, has two separate wedges, one over red and one over green. Since it is symmetric, it balances at the center. It represents a neutral gray of intermediate brightness. So does the #10. The #14 has pie over every sector except the red one. It is bright and balances on a line toward forest green. It gives a light, somewhat bluish green.

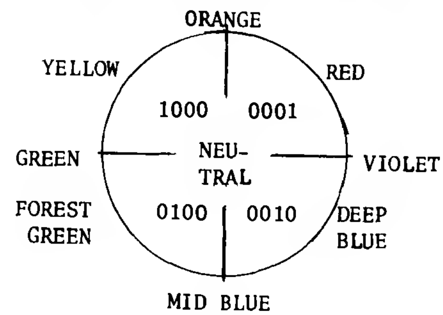



Figure 1.

Color circle shows relations of color to color number bit position.

A diagram representing the relations of all the colors is given in Figure 2. Each of the one, two and three bit numbers form planes, each corresponding to a color circle. One can think of these positions as points in space, with brightness increasing with vertical position and horizontal planes representing color circles of differing brightness.

The colors of the Apple are thus coded by the bit patterns of the numbers representing them. You can think of them as additive combinations of red, dark blue, dark green and brown, where adding two colors is represented by ORing the two numbers representing them. Subtractive combination can be represented by ANDing the light colors, pink, yellow, light green and light blue. The more bits set in a number, the brighter; the fewer, the darker. The bit patterns for 5 and 10 have no 3.58 MHz component and so generate a neutral tone. At a boundary between 5 and 10 however, this pattern is disturbed and two bits or spaces adjoin. Try the following program which has only grays displayed:

```
10 GR
20 FOR I = 0 TO 9
30 COLOR = 5
40 HLIN 0,39 AT 2*I
50 VLIN 20,39 AT 2*I
60 VLIN 20,39 AT 2*I+21
70 COLOR = 10
80 HLIN 0,39 AT 2*I + 1
90 VLIN 20,39 AT 2*I + 1
100 VLIN 20,39 AT 2*I + 20
110 NEXT I
120 RETURN
```

The top half of the display has HLIN's, alternating 5 and 10. The bottom half has VLIN's, alternating 5 and 10. What do you see? The bit pattern for a number is placed directly on the video signal, with the four bits occupying one color carrier period. When two bits adjoin at a

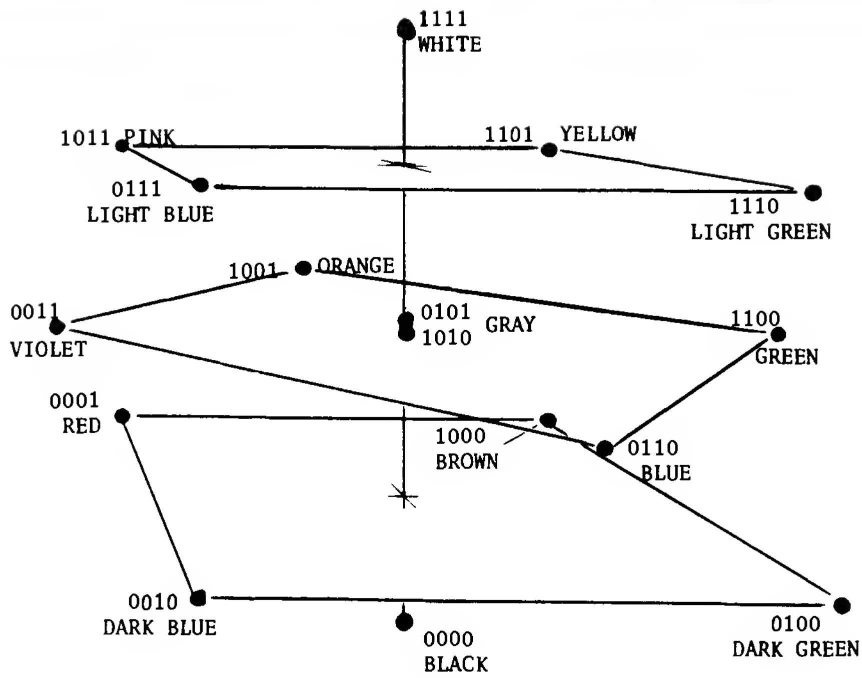



Figure 2.

Color space locations of the Apple II colors. Each horizontal plane forms a color circle of different brightness.

5,10 boundary, a light band is formed. When two spaces adjoin, a dark band is formed. The slight tints are due to the boundaries having some color component. Changing the 5,10 order reverses this tint.

Now is perhaps a good time to consider just how large a 3.58 MHz period is. The Apple text is generated with a 5x7 dot matrix, a common method of character generation. These same dots correspond to individual bits in the high resolution display memory. One dot is one-half of a 3.58 MHz period and corresponds to a violet (#3) or green (#12) color signal. This is why the test is slightly colored on a color TV and the high resolution display has two colors (other than black and white), green and violet. (But you can make others, due to effects similar to those seen in the BASIC program above.)

(The design of color TV has further implications for the display. The video black and white signal is limited to about 4 MHz, and many sets drop the display frequency response so that the color signal will not be obtrusive. A set so designed will not resolve the dots very well and will produce blurry text. Some color sets have adjustments that make the set ignore the color signal. Since the color signal processing involves subtracting and adding portions of the signal, avoiding this can sometimes improve the text resolution. Also reducing the contrast especially and the brightness somewhat can help with text material.)

The color TV design attempts to remove the color carrier from the picture (after duly providing the proper color), but you may be able to see the signal as 3 or 4 fine vertical lines per color block. They should not be apparent at all in the white or black or either gray (except possibly on a high resolution monitor).

## Tan is Between Brown and White

This section presents a brief application of the concepts of the relationships in color space of the Apple colors. Many of you, I suspect, are regular readers of Martin Gardner's "Mathematical Games" column in Scientific American. I strongly recommend it to those of you who have not already been introduced. It publicized "Life" (MICRO 5:5) and motivated "Applayer" (MICRO 5:29), and was the motivation for this program. There's a lot of gold in the mine yet.

In April, the column discussed the aesthetic properties of random variations of different kinds. To summarize briefly, three kinds are:

WHITE — Each separate element is chosen randomly and is independent of every other element. Called "white" because a frequency spectrum of the result shows all frequencies occur equally, a qualitative description of white light.

BROWN — Each separate element is the previous element plus a randomly chosen deviation. Called "brown" because Brownian montion is an example.

1/F — So called because of its frequency spectrum, intermediate between "white" and "brown".

The column presented arguments, attributed to Richard Voss, that 1/f variations are prevalent and aesthetically more satisfying than "white" (not enough coherence) or "brown" (not enough variation). An algorithm was given for generating elements with 1/f random variations. Briefly, each element is the sum of N terms (three, say). One term is chosen randomly for each element. The next is chosen randomly for every ot-

her element. The next is chosen randomly for every fourth element, and so forth.

With the Apple, one can experiment with these concepts aurally (hence Applayer) and visually with the graphic displays. Color is a dimension that was not discussed much in the column. This section presents an attempt to apply these concepts to the Apple display.

Most of us know what "white" noise is like on the Apple display. An exercise that many try is to choose a random point, a random color, plot and repeat. For example:

```
10 GR
20 X = RND(40)
30 Y = RND(40)
40 COLOR = RND(16)
50 PLOT X,Y
60 GOTO 20
```

Dispite the garish display that results, this is a "white" type of random display. Except for all being within certain limits, the color of one square has no relationship to that of its neighbors and the plotting of one square tells nothing about which square is to be plotted next.

To implement the concept of "1/f", I used the following:

1. X and Y are each the sum of three numbers, one chosen randomly from each plot, one every 20 plots and the third every 200.

2. A table of color numbers was made (DIM(16) in the program) so that color numbers near each other would correspond to colors that are near each other. The choice given in the program satisfies the following restrictions:

   a. Adjacent numbers are from adjacent planes in Figure 2.

   b. No angular change (in the color planes) is greater than 45 degrees between adjacent numbers.

3. The color number is the same for 20 plots and then is changed by an amount chosen randomly from -2 to +2. This is a "brown" noise generation concept. However, most of the display normally has color patches that have been generated long before and hence are less correlated with those currently being plotted. I'll claim credit for good intentions and let someone else calculate the power spectrum.

4. Each "plot" is actually eight symmetric plots about the various major axes. I can't even claim good intentions here; it has nothing to do with 1/f and was put in for a kaleidoscope effect. Those who are offended and/or curious can alter statement 100. They may wish then to make X and Y the sum of more than three terms, with the fourth and fifth chosen at even larger intervals.

The program follows. A paddle and push buttons are used to control the tempo and reset the display. If your paddle is not connected, substitute 0 for PDL(0).

```
>LIST
    1 DIM A(16):A(1)=0:A(2)=2:A(3
      )=6:A(4)=7:A(5)=3:A(6)=1:A(
      7)=5:A(8)=11
    2 A(9)=9:A(10)=8:A(11)=10:A(12
      )=13:A(13)=15:A(14)=14:A(15
      )=12:A(16)=4
   10 GOTO 3000
  100 PLOT X,Y: PLOT 38-X,Y: PLOT
      X,38-Y: PLOT 38-X,38-Y: PLOT
      Y,X: PLOT 38-Y,38-X: PLOT Y,
      38-X: PLOT 38-Y,X
  110 RETURN
  120 Z=16
  125 L= RND (5)-2
  130 U= RND (9):V= RND (9)
  147 FOR B=1 TO 10
  150 R=U+ RND (9):S=V+ RND (9)
  155 IF PEEK (-16286)>127 THEN GR

  160 K=K+L: IF K>16 THEN K=K-Z
  165 IF K<0 THEN K=K+Z
  170 COLOR=A(K)
  180 Q=( PDL (0)/2) ^ 2
  190 FOR I=-Q TO Q: IF PEEK (-16287
      )>127 THEN 200: NEXT I
  200 FOR I=1 TO 20
  210 X=R+ RND (6):Y=S+ RND (6): GOSUB
      100: NEXT I
  220 NEXT B
  230 GOTO 120
 1010 K=1:L=5
 1020 Z=16
 2000 GOTO 120
 3000 GR : CALL -936
 3010 PRINT "PADDLE 0 CONTROLS PATTERN
       SPEED"
 3020 PRINT "USE BUTTON 0 TO GO AT ONC
      E TO HI SPEED"
 3030 PRINT "HOLD BUTTON 1 TO CLEAR SC
      REEN"
 3040 GOTO 1010
 9000 END

>CALL 858
```

DARRELL'S APPLEWARE HOUSE

We are the APPLE experts when it comes to software. We are professionals and not just hobbyists. Data Processing is our business.

Most programs are done in Integer Basic to allow user modifications. The following programs require 20K or more of memory. All programs use parallel port printers.

BUSINESS INVENTORY ($160.00 for package)
- PROGRAM 200 ($50) - Completely maintains inventory file.
- PROGRAM 205 ($20) - Fast machine language sort on Part No.
- PROGRAM 210 ($50) - Prints sales slips, updates inventory file.
- PROGRAM 220 ($50) - Generates reorder report by manufacturer code.

APPLEDITOR ($50) - A word processor that takes care of all your letter and document needs. In two versions, 39 characters and 79 characters.
APARTMENT RENTAL PROGRAM - Prints bill and labels. Maintains arrears for each unit.
MACHINE LANGUAGE SORT FOR THE FOLLOWING PROGRAMS ($20)
UNIVERSAL DATABASE ($60) - You define your database once for each use you have in mind.
DAILY CALENDAR ($50) - Search your future or past appointments.
HOME IMPROVEMENT FILE ($50) - Store all your improvements on file for future.
HOME INVENTORY FILE ($50) - Store all your home furnishings on tape for insurance purposes.
EXPENSE ACCOUNT FILE ($50) - Maintain all your travel, meals and business or personal expenses on tape.
VENDOR FILE ($50) - Store all your vendors on file.
FILING SYSTEM CROSS REFERENCE FILE ($50) - Now you can find everything in your files.
MACHINE LANGUAGE SORT FOR ANY RECORD UP TO 255 CHARACTERS ($20)
GAMES: CAR RACE PROGRAM IN HIGH RESOLUTION GRAPHICS ($7.50)
BINGO FOR 36 PLAYERS ($10) - Uses printer to print Bingo cards.

For further information about above programs, send $1.00 for postage and handling to:

DARRELL'S APPLEWARE HOUSE
17638 157th Avenue, S.E.
Renton, Washington 98055

No C.O.D. Allow two weeks for personal check to clear. Washington residents add 5.4% sales tax. For orders under $100.00 please add $2.00 for shipping and handling. Dealer inquiries welcome.

## 6502 BIBLIOGRAPHY
## PART V

William Dial
438 Roslyn Avenue
Akron, OH 44320

335. Smith, Stephen P. "6502 Disassembler Fix", DDJ 3, No. 23, Issue 3, Pg 3 (March 1978)
ROR and ROL instructions were omitted in the previously published disassembler - DDJ 3, Issue 1. This offers a simple fix.

336. KIM-1 User Notes, Issue 9/10, (January - March 1978)
Butterfield, Jim "Dicey" page 17. A program to roll up to six dice.
Butterfield, Jim "Teaser" page 17. Jumbo version of Bob Albrecht's "Shooting Stars".
Lewart, Cass "Correction for Lancaster's TVT" page 20.
Oliver, John P. "Comments and Corrections for SUPERDUMP/LOAD" pg 21.

337. Quosig, Karl and Susan "Input/Output", Personal Computing 2, No. 4, pg 8 (April 1978).
Comments on PET problems.

338. Bishop, Robert J. "Rocket Pilot", Kilobaud No. 13, pg 90 (Jan. 1978)
And interactive game for the Apple II.

339. OSI-Small Systems Journal 2, No. 1 (January-February 1978)
Anon. "What's a USR Function". Via the USR function, one can have a 6502 BASIC program which works in conjunction with one or several machine code programs.
Anon. "Quickie". A 6502 BASIC program for converting decimal to binary numbers.
Glasser, Daniel "Chessboard". Program in 6502 BASIC for a computer chessboard which moves pieces and displays the new board. Not a chess program.
Anon. "DOS CNTRL". A BASIC program to perform transfers to or from OSI's new hard disk drive.
Anon. "Track Zero Writer". A Machine language program to modify track zero.
Anon. "9 Digit BASIC". A concise method for modifying OSI 9 Digit BASIC for an end-user 9 Digit BASIC.
Anon. "OS-65U Performs". A description of a new system said to be a new standard for microcomputer operating systems.
Anon. "500/510 Breakpoint Utilities". A breakpoint program.
Anon. "510 Tracer". A tracer program which prints a disassemble of the next instruction to be executed.

340. Bishop, Robert J. "Fiendish New QUBIC Program", 73 Magazine, No. 209, pg 78 (Feb 1978).
An attempt at producing an improved version of the original Qubic program.

341. Rosner, Richard "Daddy, Is It The PET?", ROM 1, No. 9, pg 26 (Mar/April 1978)
Description of many features and operations of the PET, including many "how to" instructions.

342. Bishop, Robert J. "LOGAN - A Logic Circuit Analysis Program", Interface Age 2, No. 6, pg 128 (May 1977). An Apple I BASIC program for analyzing networks of logic gates.

343. Bishop, Robert J. "Apple Star Trek", Interface Age 2, No. 6, pg 132 (May 1977).
Star Trek written in Apple I BASIC.

344. Chamberlin, Hal "Microcomputer Input/Output", Popular Electronics 13, No. 5, pg 86 (May 1978).
Comments on the KIM's memory-mapped I/O system.

345. Peoples Computers 6, No. 6 (May/June 1978)
Johnson, Ralph "Letters". The University of California at San Diego plans a Pascal system for the 6502.
Cole, Phyllis "Apple II". A review of this 6502 based micro.
Voros, Todd L. "Sketchcode". A technique to minimize errors and simplify the process of debugging. Listed in 6502 assembly code.
Offen, Dave "Kaleidoscope". A continuously running graphics program for the PET.
Hofheintz, M. C. "Tiny GRAPHICS". A short graphics program for the PET.

346. Gordon, H. T. "Editha", DDJ 3, Issue 5, No. 25, pg 34 (May 1978). A revision of the Fylstra KIM-1 Editor program "SWEETS" published in BYTE.

347. Tullock, Michael "PET Files", Personal Computing 2, No. 5, pg 20 (May 1978). Things your user's manual never told you about PET. How to use files.

348. O'Reilly, Francis J. "Instruction Search", Byte 3, No. 5, pg 153 (May 1978). Discussion of 6502 op code 27 and the search for other as yet undefined instructions.

349. Carpenter, Charles R. "Tiny BASIC Shortcuts", Kilobaud, Issue 18, pg 42 (June 1978). Suggests methods to expand the capabilities of Tom Pittman's Tiny BASIC for the 6502.

350. O'Haver, T. C. "More Music for the 6502", Byte 3, No. 6, pg 140 (June 1978). A music composition and generation program.

351. O'Haver, T. C. "Audio Processing with a Microcomputer", Byte 3, No. 6, pg 166 (June 1978). Adding a virtual tape loop. Uses a 6502 processor.

352. Eaton, John "Low Cost Keyboard - II", 73 Magazine, No 213, pg 100 (June 1978). Part II of an article on the low-cost keyboard. Software is designed around the 6502.

353. Swindle, David "A Sensible Expansion: Atwood Memory for your KIM", Kilobaud, Issue 19, pg 60 (July 1978). Description of a low cost method to add memory to KIM.

354. MICRO, Issue 4 (April/May 1978)
- Carpenter, C. R. "Variables Chart". Chart to layout and keep track of string and numerical variables for Apple II Applesoft BASIC.
- Floto, Charles "The PET Vet Examines Some BASIC Idiosyncrasies". Includes suggestions and modifications for a Mailing List Program by Richard Rosner.
- DeJong, Marvin L. "A Complete Morse Code Send/Receive Program for the KIM-1". Converts ASCII from a keyboard to a Morse code digital signal and also converts a Morse code digital signal to an ASCII code for display on a video system.
- O'Brien "PET Software from Commodore". New selected Application notes from Commodore.
- Floto, Charles "Early PET-Compatible Products". A review of several new accessories for the PET.
- Rowe, Mike "The MICRO Software Catalog". A continuing catalog of software available for 6502 based systems.
- Carpenter, C. R. "Apple II Printing Update". Updated information and modifications of the system described previously in MICRO No. 3.
- Chamberlin, Hal "Standard 6502 Assembly Syntax?". A plea for standardization.
- Rowe, Mike "A Worm in the Apple". Discussion of some problems encountered in interfacing the Apple to other devices such as the 6820 PIA.
- Jenkins, Gerald C. "A KIM Beeper". A short blast or two of audio for load errors, end-of-line, etc.
- Auricchio, Rick "An Apple II Programmer's Guide". Some of the previously undisclosed details of the Apple Monitor.

355. O'Connor, Clint "Book Review: Programming a Microcomputer: 6502", Kilobaud, Issue 20, pg 8 (August 1978). A very favorable review of Caxton C. Foster's book.

356. Grossman, Rick "KIM Plus Chess Equals Microchess", Kilobaud, Issue 20, pg 74 (August 1978). A challenging game of Chess can be played in KIM's 1K of memroy using MicroChess by Peter Jennings.

357. Palenik, Les "FINANC - A Home/Small-Business Financial Package", Kilobaud, Issue 20, pg 84 (August 1978). Programs include Calculations on investments, Depreciation, Loans, etc.

358. Braun, Ludwig "Commodore PET", Creative Computing 4, No. 4, pg 24 (July/August 1978)

359. Creative Computing 4, No. 4 (July/August 1978).
- Braun, Ludwig "Commodore Pet". An equipment profile which stresses the value of the PET as a teaching machine.
- North, Steve "Apple II Computer". An equipment profile points out that the Apple is not a machine for the classroom or for the S-100 hardware buff but is one of the most versatile micros on the market.
- Dawkins, Gary D. "High-Resolution Graphics for the Apple II". Allows user to draw a shape in high-resolution graphics mode from the keyboard.
- Ahl, David H. "Atari Video Computer System". An equipment profile of a 6505 based programmable game system.

360. MICRO, Issue 5 (June/July 1978)

Covitz, Frank H. "Life for your PET". LIFE written in machine language for the PET.

Rockwell International ""Rockwell's New R6500/1". The 6500/1 is a single chip NMOS microcomputer, 1 or 2 MHz, fully compatible with the 6500 family.

De Jong, Marvin L. "6502 Interfacing for Beginners: Address Decoding I". The first installment in a continuing series.

Rowe, Mike "Half a Worm in the Apple". More on the controversy on interfacing the Apple to PIA's. See also EDN May 20, 1978.

Sander-Cederlof, Bob "A Slow List for Apple BASIC". Program slows down the list process so it can be more easily reviewed.

Rowe, Mike "The Micro Software Catalog: II". The second part of this continuing series.

Synertek Inc. "Synertek's VIM-1". A good description of the many features of the 6502 based VIM-1. Similar to and compatible with KIM-1 with some new features.

Suitor, Richard F. "Applayer Music Interpreter". A music interpreter written in 6502 assembly language for the Apple, but can be used on other 6502 systems.

Dial, William "6502 Bibliography - Part IV". The fourth part of the continuing bibliography of the 6502 literature (of which this is the fifth part!).

Williams, J. C. "A Block Hex Dump and Character Map Utility Program for the KIM-1". A fully relocatable utility program which will dump a specified block of memory from a KIM to a terminal in several formats.

Rockwell International "Rockwell's AIM is Pretty Good". Rockwell's AIM 65 is an assembled versatile microcomputer system on one board plus keyboard. It has a 20-character display and a 20-character thermal printer, 4K ROM monitor, 1K RAM expandable on board to 4K. Application and Expansion connectors are fully KIM-1 compatible. TTY and Audio Cassette, DEBUG/MONITOR/ ROM or EPROM on board up to 16K. 8K BASIC will be available in ROM.

Carpenter, Chuck "Apple II Accessories and Software". Items reviewed include a renumber and append program, a serial interface board, a MODEM, Applesoft II, and the "APPLE II BASIC Programming Manual.

McCann, Michael J. "A BASIC 6502 Disassembler for Apple and PET". Accepts machine language -object code- and produces a symbolic representation that resembles an assembly listing. Originally written in Commodore BASIC, it will work with Applesoft BASIC as well.

## PROGRAMMING A MICRO-COMPUTER: 6502

by Caxton C. Foster

(Reviewed by James R. Witt, Jr.)

For those of you in the computing world who have recently purchased or constructed a microcomputer based on the 6502 microprocessor (the KIM-1 fits this description) and can't put it to reasonably practical use, then perhaps your headaches are over! Programming a Micro-Computer: 6502 by Caxton C. Foster may be exactly what you need to halt your frustrations. Foster presents the reader with a combination of reference manual for programming and an introduction to 6502 systems, specifically using the KIM-1 as a model.

The motivation behind Foster's work is practicality. Right from the beginning of the first chapter a hypothetical situation is introduced, circumstances that one might face in the course of an average day, and the microcomputer is suggested as a solution. Initially, a simple problem is introduced, a problem one would not expect a computer to solve due to its simplicity. Yet, this enables the reader to grasp the basic operation of running an uncluttered program successfully. Possible reasons as to why a certain program fails are provided to lessen confusion.

With successful completion of one program, the author wastes no time moving on to new situations. This may seem somewhat fast and confusing to those who greet micros as a totally new experience. Yet the situations do become more interesting and more challenging to solve by computer software. Such programs include: "Keybounce", "A Combination Lock", and "Digital Clock" among others. Several of these programs are completely legitimate and fully operable.

As noted before, Foster moves at a swift pace. At certain points, various instructions and KIM-1 anatomy are condensed into a mere page or two. Basic understanding of digital electronics is assumed often and may be required before fully digesting some of this material. These two minor weaknesses may tend to boggle the mind of the newcomer and hinder his comprehension of the purpose of programming and its make-up.

Suggestions: For those who are newcomers to the "sport" of computing and digital electronics, you may want to consider some other preliminary instructions BEFORE undertaking this book. If you have some sense of digital, but little knowledge of micros, you should tackle it, but should make notes of important items the first time through each chapter, and then reread the chapter to pull the odds and ends together. If you have written simple programs but have an appetite for more complex proglem-solving, then **Programming A Micro-Computer: 6502** will be a definite aid and resource in satisfying your hunger.

**Programming A Micro-Computer**: 6502, by Caxton C. Foster, published by Addison-Wesley, 1978.

## MICRO

### SUBSCRIPTION AND RENEWAL INFORMATION

If you are a subscriber to MICRO, then the code following your name on the mailing label is the number of the last issue your current subscription covers. If your code is 06, then this is your last issue. MICRO will NOT send out renewal notices. So, if your number is coming up, get your subscription renewal in soon. and, please check your label for correct address and notify us of any corrections or changes.

MICRO is currently published bi-monthly. The first issue was OCT/NOV 1977. The single copy price is $1.50. Subscriptions are $6.00 for six issues in the USA. Six issue subscriptions to other countries are listed below.
[Payment must be in US $.]

| | |
|---|---|
| Surface: Canada/Mexico | $7.00 |
| All other countries | $8.00 |
| Air Mail: Europe | $14.00 |
| South America | $14.00 |
| Central America | $12.00 |
| All other countries | $16.00 |

Issues #1, 2, 3, 4, and 5 are available while the supply lasts. The price is $1.50 per copy - USA, Canada or Mexico. Other countries add $.50 per copy surface or $1.25 per copy air mail.

Name: . . . . . . . . . . . . . . . . . . . . . .

Addr: . . . . . . . . . . . . . . . . . . . . . .

City: . . . . . . . . . . . . . . . . . . . . . .

State: . . . . . . . . . . Zip: . . . . . . .

Country: . . . . . . . . . . . . . . . . . . . .

Amount: $ . . . . . . . . Start MICRO #: . . .

Back Issues: . . . . . . . . . . . . . . . . .

Your name and address will be made available to legitimate dealers, suppliers, and other 6502 interests so that you may be kept informed of new products, current developments, and so forth - unless you specify that you do not wish your name released to these outside sources.

Send payment to:

MICRO, P.O. Box 3, S. Chelmsford, MA 01824, USA

### READER FEEDBACK

With this sixth issue of MICRO, we come to the end of MICRO's first year. We are quite pleased with the growth of MICRO, with the support we have received from authors and advertisers, and with the generally positive feedback from our readers. While it is always nice to read "love letters", we would like to get some specific information about you and your interests in the 6502 world. Please take a few minutes to answer the following questions. Your answers will very definitely effect the future course of MICRO.

1. Please describe your current 6502 based equipment in detail: type, amount of memory, and so forth:

2. Describe products you would like to purchase in the next year, whether or not they currently exist, and what you would consider a reasonable price:

3. Describe the uses you have or foresee for your 6502 based equipment:

4. What kind of articles do you want to see in MICRO:

5. Assuming the size stayed the same, would you like to see MICRO published monthly?

6. The current printing format of MICRO - the heavy stock and three hole punching - costs more than a standard magazine format. It was designed so that readers could take the journal apart and save article of interest in notebooks. We will continue this format if enough readers feel strongly about it. Please circle one:

| | | | |
|---|---|---|---|
| Keep Format or Else! | Prefer Current Format. | Don't Really Care. | Prefer Normal Magazine Format. |

7. Please rate your skill level in micros:

| | | | |
|---|---|---|---|
| Hardware: | Beginner | Intermediate | Expert |
| Software: | Beginner | Intermediate | Expert |

8. What was your favorite MICRO article?

Thank you for taking the time. Send this sheet to:
MICRO, P.O. Box 3, S. Chelmsford, MA 01824

# PET COMPOSITE VIDEO OUTPUT

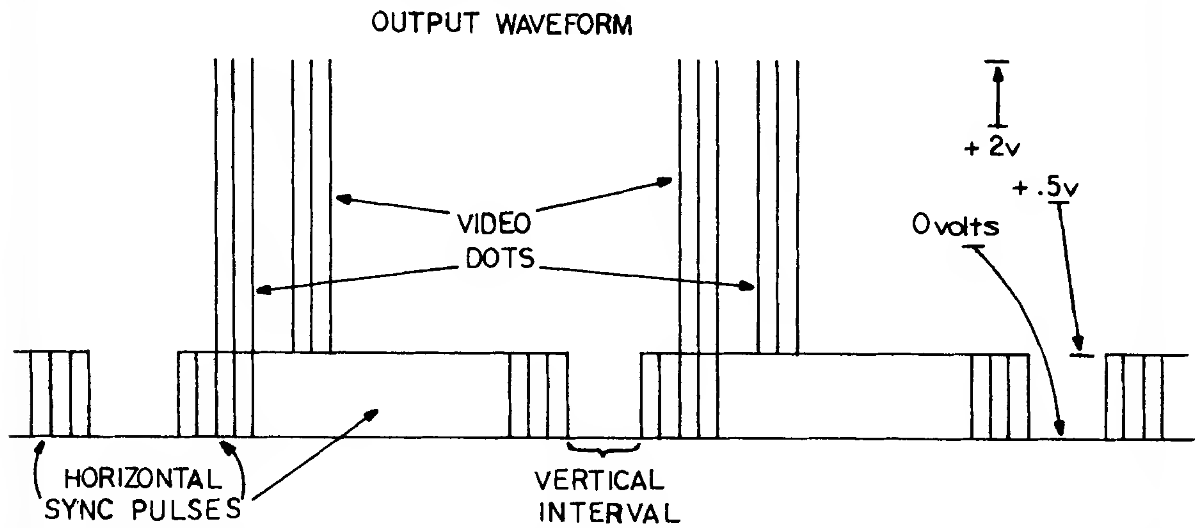
Cal E. Merritt
R. 1, 4 Richfield Lane
Danville, IN 46122

I used one of the existing PET 5 volt sources. The easiest way to steal the video and drives is to carefully scrape clean the foils next to the monitor plug and tack solder a twisted pair to each signal and to the closest ground buss. Other variations would work equally well.

To avoid metal shavings and such falling on the main board, I removed the back cover from the monitor (Power OFF) and mounted a BNC jack two inches to the right of the brightness control

The circuit is very simple and can be put together with a wire wrap tool in a few minutes.

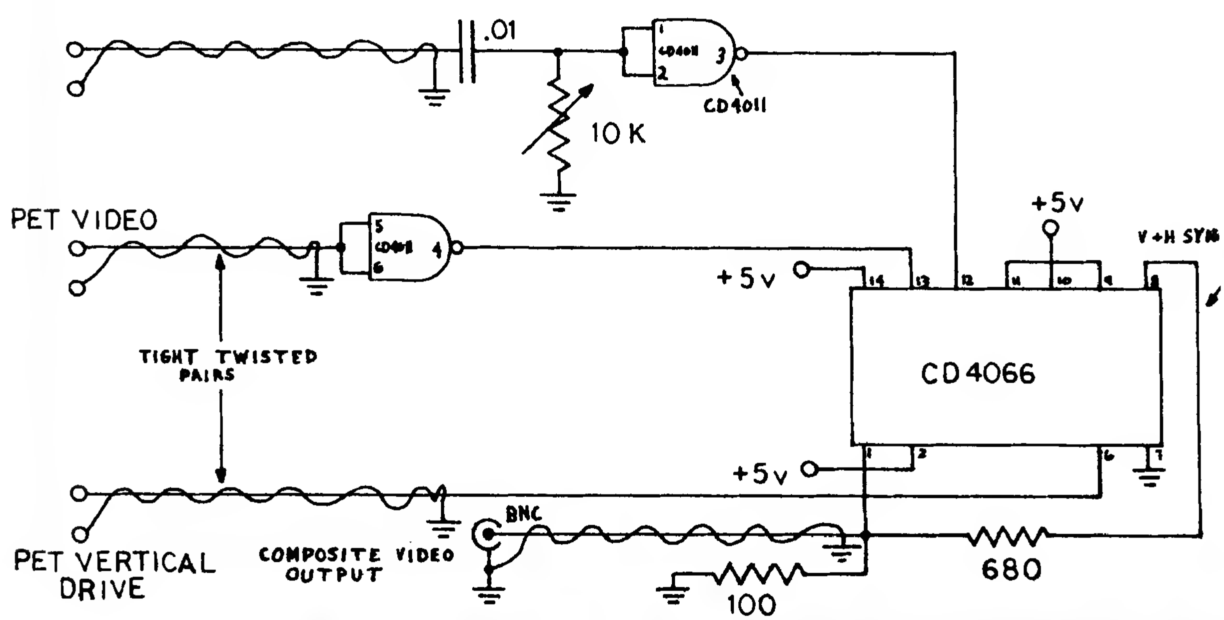
Video monitors seem very tolerant and the two units I have used work fine. The only problem encountered was in attempting to do all white screen or very dense graphics which caused sync tear in one of the monitors. Normal or dense listings worked well.




and fed it with a twisted pair. I mounted the board under one of the bolts that hold the monitor to the main chassis and attached the drive twisted pairs to the existing ones for the monitor.

This circuit provides composite video output from the PET. I have used the output to drive two different video monitors with good success.

All three monitors I tried worked with this video output. The appearance of the video will be a function of the quality of the monitor. Some of the scrapped out commercial units available with the 10MHz and more bandwidths look excellent with the PET video. I have had a number of people comment that my 12" commercial monitor looks better than the built-in unit. The add-on does not alter the existing PET display in any way.



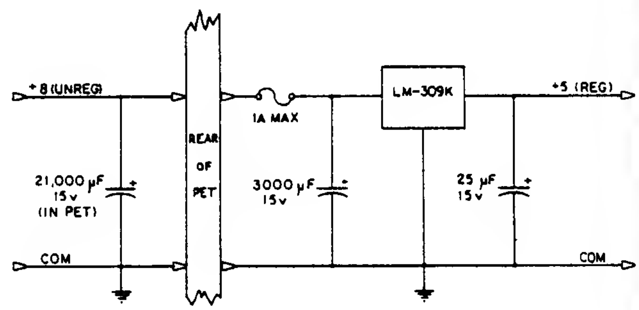
# POWER FROM THE PET

Karl E. Quosig
2038 Hartnell Street
Union City, CA 94587

It is by now well known that the PET has no source of power for use outside of itself. The only source available is at the second Cassette Interface. This +5 VDC line will not source very much current; in fact, it will not even run a second cassette recorder. Also, all the +5 VDC regulators inside the PET are already running quite warm. If you want to experiment with the PET, say with the Parallel User Port (Mos Technology 6522 VIA), then where do you get the power without a complicated power supply interface? The answer is simple. I found the following inside the PET. One, the bridge rectifier is good for 3 Amperes. Two, the PET draws 1.5 Amperes worst case load. Conclusion: it should be possible to get 1 Ampere out of the PET without straining a thing.

To do this, all we need to do is run a line from the + (positive) side of the PET's filter capacitor and make it available at the rear of the PET (I put a test lead jack between the Parallel and IEEE Ports). This is +8 VDC Unregulated and by attaching a 3-point Regulator (see diagram below), say at our project board, we have plenty of power for all sorts of home projects. As an example, I brought all of the Parallel User Port pinouts down a 24" ribbon cable along with the +8 VDC line to a chassis which has the +5 VDC regulator and other circuitry, and terminated this on a homebrew mother board comprised of 22-pin edgecard connectors. I can now experiment with things such as noise makers, joysticks, etc. and have plenty of power for them.

I believe this should be of great benefit for those of you who like to mess around with the hardware. **Warning #1:** If you are going to drill a hole in the PET as I did, disconnect all connectors (very, very gently) to the PET's Main Board and remove it before going to work. Clean inside thoroughly before re-installation. **Warning #2:** In your projects, do not connect inductive loads directly to any output of the PET. Inductive loads must be fully buffered.



# CLASSIFIED INDEX: MICRO 1 - 6



Note: "Mike Rowe" is a pseudonym for the MICRO Staff. All other names are presumed to be real.

# CLASSIFIED INDEX: MICRO 1 - 6

## GENERAL



Key to Type of Article

A = Application or Utility
B = BASIC Program
E = Editorial or Point-of-View
G = Game or Demonstration Program
H = Hardware Information
M = Machine or Assembly Language Program
P = Product Information
R = Reference Material
S = Software Information
T = Tutorial Material

## ADVERTISEMENTS

# APPLE INTEGER BASIC SUBROUTINE PACK AND LOAD

Richard F. Suitor
166 Tremont Street
Newton, MA 02158



The first issue of CONTACT, the Apple Newsletter, gave a suggestion for loading assembly language routines with a BASIC program. Simply summarized, one drops the pointer of the BASIC beginning below the assembly language portion, adds a BASIC instruction that will restore the pointer and SAVEs. The procedure is simple and effective but has two limitations. First, it is inconvenient if BASIC and the routines are widely separated (and is very tricky if the routines start at $800, just above the display portion of memory). Second, a program so saved cannot be used with another HIMEM, and is thus inconvenient to share or to submit to a software exchange.

The subroutine presented here avoids these difficulties at the expense of the effort to implement it. It is completely position independent; it may be moved from place to place in core with the monitor move command and used at the new location without modification. It makes extensive use of SWEET16, the 16 bit interpreter supplied as part of the Apple Monitor ROM.

To use the routine from Apple Integer BASIC, CALL MKUP, where MKUP is 128 (decimal) plus the first address of the routine. The prompt shown is "@". Respond with the hex limits of the routine to be stored, as BBBB.EEEE (BBBB is the beginning address, EEEE is the ending; the same format that the monitor uses). Several groups may be specified on one line separated by spaces or several lines. Type S after the last group to complete the pack and return to BASIC. The program can now be saved.

To load, enter BASIC and LOAD. When complete, RUN. The first RUN will move all routines back to their original location and return control to BASIC. It will not RUN the program; subsequent RUNs will.

A LIST of the program after calling MKUP and before the first RUN will show one BASIC statement (which initiates the restoration process) and gibberish. If this is done, RESET followed by CTRL C will return control to BASIC.

WARNING #1: The routine must be placed in core where it will not overwrite itself during the Pack. The start of the routine must be above HIMEM (e.g. in the high resolution display region) or $17A + 4*N + W below the start of the BASIC program, where N is the number of routines stored and W is the total number of words in all of these routines. Also, those routines that are highest in memory should be packed first to avoid overwriting during pack or restore. Otherwise it is not necessary to worry about overwriting during the restore process; only $1A words just below the BASIC program are used.

WARNING #2: Do not attempt to edit the program after calling MKUP. If editing is necessary, RUN once to unpack, then edit and call MKUP again.

The routine works as follows. It first packs the restore routine just below the BASIC program. It then packs other routines as requested, with first address and number of bytes (words). When S is given, it packs itself with the information to restore LOMEM and the beginning of the BASIC program. The first $46 words of the routine form a BASIC statement which will initiate the restoration process when RUN is typed.

If a particular HIMEM is needed by the program (e.g. for high resolution programs) it must be entered before LOADing. The LOMEM will be reset by the restoration process to the value it had when MKUP was called.

I do not have a SWEET16 assembler, hence all of those op codes are listed as tables of data. In the listing, comments indicate where constants and relative displacements are differences between labels in the routine.

Some convenient load and entry points are:

| BASO (load) | MKUP (entry) | |
|---|---|---|
| hex | hex | decimal |
| 800 | 880 | 2176 |
| A90 | B10 | 2832 |
| 104C | 10CC | 4300 |
| 2050 | 20D0 | 8400 |
| 3054 | 30D4 | 12500 |

Editor's Note: While we encourage the use and distribution of this subroutine, we do request that proper credit be given. Please place the following notice on any copies that you make:

"This PACK & LOAD Subroutine was written by: Richard F. Suitor and published in MICRO #6."

```
                  0010 :INT BASIC SUBR PACK & LOAD
                  0020 :CALL BAS0+128(DEC)
                  0030 ACCL .DL 0000
                  0040 BSOL .DL 0002
                  0050 TABL .DL 0004
                  0060 TBCL .DL 0006
                  0070 HIMS .DL 0008
                  0080 LMRT .DL 000A
                  0090 BPRG .DL 000C
                  0100 FRML .DL 000E
                  0110 NBYT .DL 0010
                  0120 BPR2 .DL 0012
                  0130 PTLL .DL 0014
                  0140 XTAB .DL 0016
                  0150 SKPL .DL 0018
                  0160 MODE .DL 0031
                  0170 YSAV .DL 0034
                  0180 PRMP .DL 0033
                  0190 LMML .DL 004A
                  0200 HIML .DL 004C
                  0210 LMWL .DL 00CC
                  0220 BBSL .DL 00CA
                  0230 JSRL .DL 00CE
                  0240 BSC2 .DL E003      BASIC
                  0250 BUFF .DL 0200
                  0260 GTNM .DL FFA7
                  0270 PBL2 .DL F94A
                  0280 COUT .DL FDED
                  0290 BELL .DL FF3A
                  0300 GTLN .DL FD67
                  0310 SW16 .DL F689
                  0320 :BASIC INST. TO RESTORE
0800  460000      0330 BAS0 .HS 46000064B101
0803  64B101
0806  0065B7      0340      .HS 0065B74C000364B2
0809  4C0003
080C  64B2
080E  020065      0350      .HS 020065382E3FB2CA
0811  382E3F
0814  B2CA
0816  007212      0360      .HS 007212B74600721F
0819  B74600
081C  721F
081E  B20001      0370      .HS B200010364B30300
0821  0364B3
0824  0300
0826  65382E      0380      .HS 65382E3FB2CB0072
0829  3FB2CB
082C  0072
082E  12382E      0390      .HS 12382E3FB2CA0072
0831  3FB2CA
0834  0072
0836  12B746      0400      .HS 12B746007215B200
0839  007215
083C  B200
083E  017203      0410      .HS 0172034DB1010001
0841  4DB101
0844  0001
                  0420 :INIT. RESTORE OP
0846  D8          0430 PTBK CLD
0847  A201        0440      LDX 01
0849  B5CA        0450 PT02 LDA *BBSL,X
084B  9502        0460      STA *BSOL,X
084D  B54C        0470      LDA *HIML,X
084F  9508        0480      STA *HIMS,X
0851  CA          0490      DEX
0852  10F5        0500      BPL PT02
0854  2089F6      0510      JSR SW16
```

SYMBOL TABLE

| | |
|---|---|
| ACCL | 0000 |
| BSOL | 0002 |
| TABL | 0004 |
| TBCL | 0006 |
| HIMS | 0008 |
| LMRT | 000A |
| BPRG | 000C |
| FRML | 000E |
| NBYT | 0010 |
| BPR2 | 0012 |
| PTLL | 0014 |
| XTAB | 0016 |
| SKPL | 0018 |
| MODE | 0031 |
| YSAV | 0034 |
| PRMP | 0033 |
| LMML | 004A |
| HIML | 004C |
| LMWL | 00CC |
| BBSL | 00CA |
| JSRL | 00CE |
| BSC2 | E003 |
| BUFF | 0200 |
| GTNM | FFA7 |
| PBL2 | F94A |
| COUT | FDED |
| BELL | FF3A |
| GTLN | FD67 |
| SW16 | F689 |
| BAS0 | 0800 |
| PTBK | 0846 |
| PT02 | 0849 |
| PT04 | 0870 |
| MKUP | 0880 |
| MK21 | 0882 |
| MK22 | 08B3 |
| MK01 | 08B4 |
| MK06 | 08CA |
| MERR | 08D1 |
| MK05 | 08DE |
| MK02 | 08E1 |
| MV51 | 08EB |
| MV52 | 08F5 |
| SM02 | 0909 |
| SM03 | 090B |
| MK09 | 090C |
| MK11 | 091A |
| MK12 | 091B |
| MK10 | 0932 |
| SM04 | 0946 |
| PTLP | 0952 |
| PLP0 | 0955 |
| PLP1 | 095A |
| PLP2 | 0966 |
| ST16 | 096A |

```
0857  105201   0520        .HS 105201    PLTP-BAS0
085A  185701   0530        .HS 185701    PLTP+5-BAS0
085D  A13767   0540        .HS A13767356736
0860  356736
0863  24B636   0550        .HS 24B636
0866  1A1100   0560        .HS 1A1100    ST16+1-PLP1
0869  BA3A     0570        .HS BA3A
086B  6733     0580        .HS 6733
086D  00       0590        .HS 00
086E  A201     0600        LDX 01
               0610   :SET LOMEM & BASIC PROG START
0870  B50A     0620   PT04 LDA *LMRT,X
0872  954A     0630        STA *LMML,X
0874  95CC     0640        STA *LMWL,X
0876  B50C     0650        LDA *BPRG,X
0878  95CA     0660        STA *BBSL,X
087A  CA       0670        DEX
087B  10F3     0680        BPL PT04
087D  6C1400   0690        JMP (PTLL)    TO RESTORE LP
               0700   :SUBR TO SET UP PACK
0880  A201     0710   MKUP LDX 01
0882  B54A     0720   MK21 LDA *LMML,X
0884  950A     0730        STA *LMRT,X
0886  B5CA     0740        LDA *BBSL,X
0888  9512     0750        STA *BPR2,X
088A  950C     0760        STA *BPRG,X
088C  B5CE     0770        LDA *JSRL,X
088E  9504     0780        STA *TABL,X
0890  B54C     0790        LDA *HIML,X
0892  9508     0800        STA *HIMS,X
0894  CA       0810        DEX
0895  10EB     0820        BPL MK21
               0830   :INIT & PACK RESTORE LP
0897  2089F6   0840        JSR SW16
089A  24B939   0850        .HS 24B939
089D  118000   0860        .HS 118000    MKUP-BAS0
08A0  22B131   0870        .HS 22B131
08A3  105201   0880        .HS 105201    PLTP-BAS0
08A6  A13218   0890        .HS A132181800 ST16-PTLP
08A9  1800
08AB  A833E3   0900        .HS A833E3
08AE  1C5000   0910        .HS 1C5000
08B1  0C42     0920        .HS 0C42      MV52-MK22
08B3  00       0930   MK22 .HS 00
08B4  A9C0     0940   MK01 LDA 0C0
               0950   :GET LIMITS & PACK PROGS
08B6  8533     0960        STA *PRMP
08B8  A900     0970        LDA 0
08BA  8531     0980        STA *MODE
08BC  2067FD   0990        JSR GTLN
08BF  8616     1000        STX *XTAB
08C1  A000     1010        LDY 00
08C3  B90002   1020        LDA BUFF,Y
08C6  C9D3     1030        CMP 0D3       S
08C8  F068     1040        BEQ MK10
08CA  20A7FF   1050   MK06 JSR GTNM
08CD  C9A7     1060        CMP 0A7       F('.')
08CF  F010     1070        BEQ MK02
08D1  98       1080   MERR TYA
08D2  AA       1090        TAX
08D3  204AF9   1100        JSR PBL2      ERROR INDICATOR
08D6  A95E     1110        LDA '^
08D8  20EDFD   1120        JSR COUT
08DB  203AFF   1130        JSR BELL
08DE  18       1140   MK05 CLC
08DF  90D3     1150        BCC MK01
08E1  E631     1160   MK02 INC *MODE
08E3  20A7FF   1170        JSR GTNM
```

```
                  1180  :A1 & A3 NOW HAVE 1ST =,A2 2D
                  1190  :SET UP MOVE TO JUST BELOW (BBSL)
                  1200  :AND LOWER BBSL
08E6  2089F6      1210       JSR SW16
08E9  011E        1220       .HS 011E       SM02-MV51
08EB  183C00      1230  MV51 .HS 183C0068326833
08EE  683268
08F1  33
08F2  B238E3      1240       .HS B238E3
08F5  839623      1250  MV52 .HS 839623D207FA
08F8  D207FA
08FB  283318      1260       .HS 2833180800
08FE  0800
0900  889688      1270       .HS 8896889688968896
0903  968896
0906  8896
0908  0B          1280       .HS 0B
0909  0CE0        1290  SM02 .HS 0CE0       MV51-SM03
090B  00          1300  SM03 .HS 00
090C  C9EC        1310  MK09 CMP 0EC        F('S')
090E  F022        1320       BEQ MK10
0910  C9C6        1330       CMP 0C6        F(CR)
0912  F0A0        1340       BEQ MK01
0914  C999        1350       CMP 99         BLANK
0916  F003        1360       BEQ MK12
0918  D0B7        1370       BNE MERR
091A  C8          1380  MK11 INY
091B  B90002      1390  MK12 LDA BUFF,Y
091E  C416        1400       CPY *XTAB
0920  B092        1410       BCS MK01
0922  C9A0        1420       CMP 0A0        BLANK
0924  F0F4        1430       BEQ MK11
0926  C98D        1440       CMP 8D
0928  F08A        1450       BEQ MK01
092A  C9D3        1460       CMP 0D3        S
092C  F004        1470       BEQ MK10
092E  C631        1480       DEC *MODE
0930  F098        1490       BEQ MK06       ALWAYS
                  1500  :PACK 1ST PART & CLEAN UP
0932  2089F6      1510  MK10 JSR SW16
0935  2132        1520       .HS 2132
0937  185201      1530       .HS 185201     PTLP-BAS0
093A  A83725      1540       .HS A83725772977
093D  772977
0940  2177        1550       .HS 2177
0942  2733        1560       .HS 2733
0944  0CAF        1570       .HS 0CAF       MV52-SM04
0946  6666        1580  SM04 .HS 6666
0948  00          1590       .HS 00
0949  A50C        1600       LDA *BPR6
094B  85CA        1610       STA *BBSL
094D  A50D        1620       LDA *BPR6+01
094F  85CB        1630       STA *BBSL+01
0951  60          1640       RTS
                  1650  :RESTORE LOOP
0952  2089F6      1660  PTLP JSR SW16
0955  613361      1670  PLP0 .HS 6133613800 GET POINT
0958  3800
095A  2089F6      1680  PLP1 JSR SW16
095D  4153F8      1690       .HS 4153F804FB
0960  04FB
0962  21D605      1700       .HS 21D605
0965  EF          1710       .HS EF         PLP0-PLP2
0966  00          1720  PLP2 .HS 00
0967  4C03E0      1730       JMP BSC2
096A  00          1740  ST16 .HS 00
                  1750       .EN
```

# A PARTIAL LIST OF PET SCRATCH PAD MEMORY

Gary A. Creighton
625 Orange Street, No. 43
New Haven, CT 06510

A function and a symbol defined:

```
DEF FN IND(LOC) = PEEK(LOC+1)*256+PEEK(LOC)
      Which specifies an indirect address in the form:LOC+1=(Page)
                                                      LOC  =(Item)
M(LOC)          specifies contents of a memory location.
```

| | |
|---|---|
| M(0) | JMP instruction |
| FN IND(1) | USR jump location |
| M(3) | Present I/O Device Number (suppress printout) |
| M(5) | POS function store |
| FN IND(8) | Arguments of commands with range 0 to 65535 (PEEK,POKE,WAIT,SYS,GOTO,GOSUB,Line Number,RAM check) |
| M(10-89) | Input Buffer |
| M(90-98) | Flags for MISMATCH, Distinguishing between similar subroutines, etc. |
| M(91) | Ignore Code Value and do direct (between quotes, etc.) |
| M(98) | (0 INPUT, 64 GET/GET#, 152 READ) Flag |
| FN IND(113) | Transfer Number pointer |
| FN IND(115) | Number pointer |
| FN IND(122) | Begin Basic Code pointer |
| FN IND(124) | Begin Variables pointer |
| FN IND(126) | Variable List pointer |
| FN IND(128) | End Variables pointer |
| FN IND(130) | Lowest String Variables pointer |
| FN IND(132) | Highest String Variables pointer |
| FN IND(134) | First Free After Strings pointer |
| FN IND(136) | Present Line Number (if M(137)=255, no line number) |
| FN IND(138) | Line Number at BREAK |
| FN IND(140) | Continue Run pointer (if M(141)=0, can't continue) |
| FN IND(142) | Line Number of Present DATA line |
| FN IND(144) | Next DATA pointer (for READ) |
| FN IND(146) | Next Data/Input After Last Comma pointer |
| M(148) | Coded 1st Character of Last Variable |
| M(149) | Coded 2nd Character of Last Variable |
| FN IND(150) | Variable pointer (all variables) |
| FN IND(152) | Variable pointer |
| M(156) | Comparison Symbol Accumulator (<=>) |
| FN IND(157) | Pointer to FN pointer |
| M(157-161) | Number Store/Work area (SQR) |
| M(163-165) | JMP (FN IND(164)) |
| FN IND(164) | Function Jump address |
| M(166-170) | Number Store/Work area (Transcendentals (not EXP) & SQR) |
| M(171-175) | Number Store/Work area (Transcendentals & SQR) |
| M(176-181) | Main Number Store/Work area |
| M(181) | Number Sign |
| M(184-189) | Secondary Number Store/Work area |
| M(192) | Length of things in Input Buffer M(10-89) or Length of things in Output Number M(256- )...other |
| M(194-217) | Subroutine: Point through code one at a time, RTS with code value in accumulator and Carry Flag Clear if ASC(0-9) 0 if end of line. Ignore Spaces. |
| FN IND(201) | Code Pointer |
| M(218-222) | Number Store/Work area (RND) |
| FN IND(224) | Screen Memory Row location |
| M(226) | Screen Column position |

| | |
|---|---|
| FN IND(227) | Move Memory (from or to) pointer |
| M(234) | Quote flag (0 end quote)(1 begin quote) |
| M(238) | Length of File name after SAVE VERIFY etc. |
| M(239) | File # |
| M(240) | I/O Option (0 read, 1 write, 2 write/EOT) |
| M(241) | Device # (0 keyboard, 1 tape#1, 2 tpae#2, 3 screen) |
| M(242) | Wraparound flag (39 single line, 79 2nd of double line) |
| FN IND(243) | Tape #1 or #2 Buffer pointer |
| M(245) | Screen Row (0 - 24) |
| FN IND(247) | Load into/ Verify from? Save into pointer |
| M(251) | Insert Counter ( INST) |
| M(256) | Minus sign or Space for Output Number |
| M(256- ) | Output Number ASC Digits til a Null (0) or Tape Read Working Storage |
| M(311?-511) | Stack area |
| M(512-514) | TI clock |
| M(515) | Only One Value per Keypush flag |
| M(516) | SHIFT flag (0 no shift, 1 shift) |
| M(517-518) | TI Update Interrupt Counter |
| M(521) or | Bit Cancel Keys |
| M(59410) | Turns bits off under the following rules: |

| BIT | KEY | DECIMAL # | |
|---|---|---|---|
| 0 | RVS | 254 | |
| 1 | | 253 | |
| 2 | space | 251 | More than one key |
| 3 | | 247 | |
| 4 | stop | 239 | may be pushed at once. |
| 5 | (none) | | |
| 6 | | 191 | Decimal # is Binary |
| 7 | | 127 | equivalent. |

| | |
|---|---|
| M(523) | VERIFY/LOAD flag (0 LOAD, 1 VERIFY) |
| M(524) | ST Status |
| M(525) | Key Pushed Counter (MOD 10) |
| M(526) | RVS flag (0 RVS off, 1 RVS on) or any key pushed) |
| M(527-536) | Input Run Buffer (keys stored during a RUN |
| FN IND(537) | Interrupt Vector (normally at: Store Keypush |
| FN IND(539) | BRK instruction Vector (User loaded) in Input Run Buffer) |
| M(547) | Keyboard Input Code (Stays equal to Input code til finger off key, Matches up one to one with M(59228-59307) which is Keyboard Input Code to ASC Code Table) |
| M(548) | Blink Cursor flag (if 0 (no key pushed)) |
| M(549) | Cursor Blink Duration counter (20 interrupts) |
| M(550) | Screen Value of Input Char. when Cursor moves on |
| M(551) | Insure no Cursor Breadcrumbs left behind |
| M(553-577) | Screen Page Array / single or double Line flags |
| M(578-587) | File # of one of 10 files |
| M(588-597) | Device # of one of 10 files |
| M(598-607 | I/O option one of 10 files |
| M(608) | Input from screen/Input from keyboard flag |
| M(610) | Number of Open Files |
| M(611) | Device Number of Input Device (0 keyboard normally) |
| M(612) | Device Number of Output Device (3 screen normally) |
| M(616) | Tape Buffer Item Counter |
| M(634-825) | Tape #1 Buffer area |
| M(826-1023) | Tape #2 Buffer area |